Professora Me. Márcia Cristina Dadalto Pascutti

ENGENHARIA DE SOFTWARE

GRADUAÇÃO

CURSO

MARINGÁ-PR

2012

ENGENHARIA DE SOFTWARE

Professora Me. Márcia Cristina Dadalto Pascutti

APRESENTAÇÃO DO REITOR

Viver e trabalhar em uma sociedade global é um grande desafio para todos os cidadãos. A busca por tecnologia, informação, conhecimento de qualidade, novas habilidades para liderança e solução de problemas com eficiência tornou-se uma questão de sobrevivência no mundo do trabalho.

Cada um de nós tem uma grande responsabilidade: as escolhas que fizermos por nós e pelos nossos fará grande diferença no futuro.

Com essa visão, o Cesumar – Centro Universitário de Maringá – assume o compromisso de democratizar o conhecimento por meio de alta tecnologia e contribuir para o futuro dos brasileiros.

No cumprimento de sua missão – "promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária" –, o Cesumar busca a integração do ensino-pesquisa-extensão com as demandas institucionais e sociais; a realização de uma prática acadêmica que contribua para o desenvolvimento da consciência social e política e, por fim, a democratização do conhecimento acadêmico com a articulação e a integração com a sociedade.

Diante disso, o Cesumar almeja ser reconhecido como uma instituição universitária de referência regional e nacional pela qualidade e compromisso do corpo docente; aquisição de competências institucionais para o desenvolvimento de linhas de pesquisa; consolidação da extensão universitária; qualidade da oferta dos ensinos presencial e a distância; bem-estar e satisfação da comunidade interna; qualidade da gestão acadêmica e administrativa; compromisso social de inclusão; processos de cooperação e parceria com o mundo do trabalho, como também pelo compromisso e relacionamento permanente com os egressos, incentivando a educação continuada.

Professor Wilson de Matos Silva
Reitor

Caro aluno, "*ensinar não é transferir conhecimento, mas criar as possibilidades para a sua produção ou a sua construção*" (FREIRE, 1996, p. 25). Tenho a certeza de que no Núcleo de Educação a Distância do Cesumar, você terá à sua disposição todas as condições para se fazer um competente profissional e, assim, colaborar efetivamente para o desenvolvimento da realidade social em que está inserido.

Todas as atividades de estudo presentes neste material foram desenvolvidas para atender o seu processo de formação e contemplam as diretrizes curriculares dos cursos de graduação, determinadas pelo Ministério da Educação (MEC). Desta forma, buscando atender essas necessidades, dispomos de uma equipe de profissionais multidisciplinares para que, independente da distância geográfica que você esteja, possamos interagir e, assim, fazer-se presentes no seu processo de ensino-aprendizagem-conhecimento.

Neste sentido, por meio de um modelo pedagógico interativo, possibilitamos que, efetivamente, você construa e amplie a sua rede de conhecimentos. Essa interatividade será vivenciada especialmente no ambiente virtual de aprendizagem – AVA – no qual disponibilizamos, além do material produzido em linguagem dialógica, aulas sobre os conteúdos abordados, atividades de estudo, enfim, um mundo de linguagens diferenciadas e ricas de possibilidades efetivas para a sua aprendizagem. Assim sendo, todas as atividades de ensino, disponibilizadas para o seu processo de formação, têm por intuito possibilitar o desenvolvimento de novas competências necessárias para que você se aproprie do conhecimento de forma colaborativa.

Portanto, recomendo que durante a realização de seu curso, você procure interagir com os textos, fazer anotações, responder às atividades de autoestudo, participar ativamente dos fóruns, ver as indicações de leitura e realizar novas pesquisas sobre os assuntos tratados, pois tais atividades lhe possibilitarão organizar o seu processo educativo e, assim, superar os desafios na construção de conhecimentos. Para finalizar essa mensagem de boas-vindas, lhe estendo o convite para que caminhe conosco na Comunidade do Conhecimento e vivencie a oportunidade de constituir-se sujeito do seu processo de aprendizagem e membro de uma comunidade mais universal e igualitária.

Um grande abraço e ótimos momentos de construção de aprendizagem!

Professora Gislene Miotto Catolino Raymundo

Coordenadora Pedagógica do NEAD- CESUMAR

APRESENTAÇÃO

Livro: ENGENHARIA DE SOFTWARE

Professora Me. Márcia Cristina Dadalto Pascutti

Prezado(a) acadêmico(a), é com muito prazer que apresento a você o livro de Engenharia de Software I. Sou a professora Márcia Cristina Dadalto Pascutti, autora deste material e, pode ter certeza, que o mesmo foi preparado com carinho especial para que você possa entender o que esta disciplina pode te trazer de benefício ao longo de sua vida como desenvolvedor de sistemas.

Ministro esta disciplina em cursos de graduação há, praticamente, dezesseis anos. Inicialmente, a disciplina era chamada de Análise de Sistemas e nos últimos anos, de Engenharia de Software, mas sempre com o mesmo objetivo, ou seja, mostrar ao aluno o processo de desenvolvimento de um software. Escrever este material foi um desafio, pois é preciso escrever de forma clara para que você, querido(a) aluno(a), possa entender o meu raciocínio. Mas foi uma experiência ótima e espero que você goste do que foi colocado aqui.

A engenharia de software possui uma gama imensa de tópicos, que não seria possível abordar em cinco unidades (como este livro está organizado), então coloquei os assuntos iniciais, sem os quais não seria possível entender todo o contexto da disciplina. Se você gostar da disciplina e quiser se aprofundar, pode consultar os livros que cito durante as unidades. Neles, com certeza, você aprenderá muitos outros assuntos e poderá ter certeza de que a engenharia de software realmente é muito importante quando tratamos de software, como um todo.

É muito importante que você não deixe de realizar as atividades de autoestudo para fixar os conceitos estudados durante a unidade. Lembre-se de que a unidade seguinte sempre está vinculada com a unidade anterior, portanto, é saudável que você entenda todo o conteúdo de uma unidade para passar para a próxima. Se ainda ficar com dúvida, procure realizar pesquisas na internet e fazer as leituras complementares indicadas. Pode ter certeza que isso o ajudará.

Este livro está organizado em cinco unidades, sendo que todas estão estreitamente relacionadas. Na unidade I apresentarei alguns conceitos referentes à disciplina. Você notará, durante a leitura das outras unidades, que esses conceitos são utilizados com frequência.

A engenharia de software surgiu da necessidade de tornar o desenvolvimento de software confiável, com etapas bem definidas, com custo e cronograma previsíveis, o que, até a época de 1968, quando o termo engenharia de software foi proposto, não acontecia. A proposta da engenharia de software, portanto, é tornar o desenvolvimento de software um processo sistematizado, no qual podem ser aplicados técnicas e métodos necessários para controlar a complexidade inerente aos grandes sistemas (SOMMERVILLE, 2007, p. 4).

Gostaria de ressaltar que software compreende, além dos programas, toda a documentação referente aos mesmos, sendo que a engenharia de software é a disciplina que trata dessa documentação. Sendo assim, apresentarei a você uma pequena parte dessa documentação, pois seria necessário mais do que um livro para tratar da documentação completa que pode ser elaborada no desenvolvimento de um software. Outro ponto importante que é necessário deixar registrado é que de nada vale uma documentação desatualizada, por isso, é importante utilizar uma ferramenta CASE para criar e manter a documentação. Uma ferramenta CASE também é um software, só que neste caso, auxilia o desenvolvedor e não o usuário final do sistema que está sendo desenvolvido.

Depois, na segunda unidade, começaremos a aplicar os conceitos já estudados e mostrarei a você que um processo de software genérico é composto de algumas etapas básicas que farão parte de qualquer modelo de processo de software. Essas etapas básicas são: i) a especificação dos requisitos do software a ser desenvolvido; ii) o projeto e a implementação do software; iii) a validação do software e, finalmente iv) a evolução do software. Nesta unidade, abordarei três modelos de processo de software, mas a literatura traz outros modelos, como, por exemplo, o Rational Unified Proccess. Meu objetivo é mostrar alguns modelos propostos pela literatura, mas, ao final dessa unidade, você poderá elaborar o seu próprio modelo, colocando as etapas na ordem que você achar melhor para a sua empresa.

A unidade III é bastante específica, tratando apenas dos conceitos relacionados a requisitos de software, já que, para o desenvolvimento de um software da forma esperada pelo cliente, é fundamental que todos os requisitos tenham sido claramente definidos. Nesta unidade, mostrarei a você o que é um documento de requisitos e porque ele é tão importante.

Um requisito deve estabelecer o que o sistema deve fazer, bem como as restrições sobre seu funcionamento e implementação, podendo ser classificado em requisito funcional e não funcional. Os requisitos funcionais dizem respeito aos serviços que o software deve fornecer,

como, por exemplo, o cadastro dos pacientes de uma clínica odontológica, a emissão de um relatório de vendas. Já os requisitos não funcionais normalmente estão relacionados às propriedades emergentes do sistema, podendo ser aplicados ao sistema como um todo. Um exemplo de requisito não funcional: o sistema deve ser desenvolvido em seis meses.

Todos esses requisitos, tanto os funcionais quanto os não funcionais, devem ser claramente definidos no documento de requisitos, pois é a partir desse documento que o sistema será modelado, projetado, implementado, ou seja, todo o desenvolvimento do sistema será baseado neste documento. Em alguns casos, o documento de requisitos pode servir como um contrato entre o cliente e a empresa desenvolvedora do software.

Para você ter uma ideia de como é um documento de requisitos, mostrarei o de uma locadora de filmes na unidade III. O exemplo é bem simples, mas contém detalhes suficientes para a continuidade do processo de desenvolvimento de software.

Então, de posse do documento de requisitos, começaremos a estudar a penúltima unidade do nosso livro, a unidade que trata da modelagem do sistema. Nesta unidade, utilizaremos os conceitos de orientação a objetos e de UML estudados na primeira unidade. A modelagem é a parte fundamental de todas as atividades relacionadas ao processo de software, sendo que, os modelos que são construídos nesta etapa servem para comunicar a estrutura e o comportamento desejados do sistema, a fim de que possamos melhor compreender o sistema que está sendo elaborado.

Representaremos estes modelos por meio de diagramas, utilizando a UML como notação gráfica. Na primeira unidade já explicamos a importância da UML e agora começaremos a utilizá-la de fato. A UML tem diversos diagramas, mas, em nosso material, trataremos somente do Diagrama de Casos de Uso e do Diagrama de Classes. A fim de auxiliar o entendimento de cada um deles, elaborei uma espécie de tutorial explicando a elaboração dos mesmos passo a passo. Isso foi feito para facilitar o seu entendimento, pois esses diagramas são os mais importantes e os mais utilizados da UML, servindo de base para os demais diagramas.

E, finalmente, chegamos à última unidade do nosso material. Essa unidade é o fechamento das etapas do processo de software, ou seja, tratará das etapas de projeto, validação e evolução de software. Assim, você compreenderá todo o processo de software com as etapas que o englobam.

O projeto de software é onde são definidas as interfaces do sistema. Você pode fazer isso já utilizando uma linguagem de programação (de preferência aquela em que você vai implementar o sistema) ou então alguma ferramenta CASE para prototipação de sistema. É importante que o usuário participe ativamente deste processo, afinal, será ele quem vai passar a maior parte do tempo interagindo com o sistema. Depois disso, o sistema pode ser implementado, ou seja, é hora de transformar todo o trabalho realizado até o momento em código fonte.

À medida que o sistema vai sendo desenvolvido, cada programa vai sendo validado pelo desenvolvedor, mas isso só não basta. É muito importante que a etapa de validação seja cuidadosamente realizada pela equipe de desenvolvimento, pois é preciso assegurar que o sistema funcionará corretamente.

Depois que o sistema é colocado em funcionamento, ele precisa evoluir para continuar atendendo as necessidades dos usuários. Em todas estas etapas é importante a aplicação das técnicas da engenharia de software.

Assim, chegamos ao final do nosso livro. Espero, sinceramente, que sua leitura seja agradável e que esse conteúdo possa contribuir para seu crescimento pessoal e profissional.

Profª. Márcia.

SUMÁRIO

UNIDADE III

REQUISITOS DE SOFTWARE

UNIDADE IV

MODELAGEM DE SISTEMAS

UNIDADE I

INTRODUÇÃO À ENGENHARIA DE SOFTWARE

Professora Me. Márcia Cristina Dadalto Pascutti

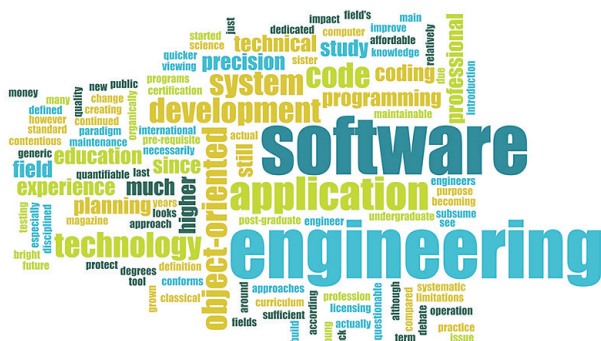
Objetivos de Aprendizagem

- Entender o que é engenharia de software e qual a sua importância.
- Estudar os conceitos básicos de orientação a objetos necessários para o entendimento da UML.
- Abordar a importância da utilização da UML para a modelagem de sistemas.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- **Conceitos básicos sobre Software e Engenharia de Software**
- **Tipos de Aplicações de Software**
- **Ferramentas CASE**
- **Conceitos Básicos de Orientação a Objetos**
- **Introdução à UML**

INTRODUÇÃO

Caro(a) aluno(a), nesta primeira unidade trataremos alguns conceitos relacionados à engenharia de software como um todo que serão fundamentais para o entendimento de todas as unidades.

O termo engenharia de software foi proposto, inicialmente, em 1968, em uma conferência organizada para discutir o que era chamado de crise de software. Essa crise foi originada em função do hardware, que nessa época tinha seu poder de processamento e memória aumentados, sendo que o software deveria acompanhar esse avanço. E o que se notou, na época, é que o desenvolvimento de grandes sistemas de maneira informal, sem seguir regras ou etapas pré-definidas, **não era suficiente.**

O software desenvolvido até então, não era confiável, não era desenvolvido dentro do tempo e custos previstos inicialmente, seu desempenho era insatisfatório e era difícil de dar manutenção ao mesmo. Os custos em relação ao hardware estavam caindo, em função de que a sua produção passou a ser em série, porém, o custo do software não acompanhava essa queda, muitas vezes, sendo aumentado.

A ideia inicial da engenharia de software era tornar o desenvolvimento de software um processo sistematizado, em que seriam aplicadas técnicas e métodos necessários para controlar a complexidade inerente aos grandes sistemas (SOMMERVILLE, 2007, p. 4).

Anotações

Apresentarei a você o conceito de software e você verá que software é muito abrangente, englobando desde os programas escritos até a documentação associada a esse software. Cabe aqui ressaltar que essa documentação deve estar sempre atualizada, pois, caso contrário, ela perde o seu sentido. Para ajudar nessa tarefa de manter a documentação em dia, podem ser utilizadas as ferramentas CASE, que também veremos nessa unidade. Um detalhe interessante é que uma ferramenta CASE também é um software.

Depois, trataremos sobre a engenharia de software como um todo. Neste livro, abordarei somente alguns tópicos da engenharia, mas, com certeza, precisaríamos de dezenas de livros como esse para esgotar esse assunto, portanto, gostaria de deixar claro, que aqui estudaremos somente uma pequena parte dessa abrangente disciplina.

Os exemplos que serão mostrados neste livro são simples para que seja mais fácil entendermos os conceitos apresentados, mas a engenharia de software pode ser aplicada a um conjunto imenso de tipos diferentes de soluções que requerem software. Você, com certeza, já deve ter utilizado um micro-ondas. Então ... quando você pressiona a tecla pipoca, é um software embutido que está sendo executado. Portanto, com a tecnologia que temos hoje à nossa disposição, é possível encontrar o software em muitos lugares.

Você vai perceber que grande parte da documentação do sistema produzido a partir da aplicação das técnicas e métodos da engenharia de software, é gráfica, ou seja, acontece através de diagramas. Como a UML (Unified Modeling Language – Linguagem de Modelagem Unificada) é a linguagem para modelagem mais utilizada atualmente, vamos também utilizá-la neste livro.

Porém, para entender qualquer diagrama da UML é necessário conhecer alguns conceitos relacionados à orientação a objetos, como, por exemplo, classe, objeto, herança. Após esses conceitos serem apresentados é que iniciaremos o estudo da UML.

Nesta unidade, veremos somente uma introdução sobre a UML. Você verá como ela surgiu

Anotações

e verá também que é uma forma de representação orientada a objetos bastante recente. Na unidade III é que estudaremos dois dos diagramas da UML, por isso é importante entender, nesta unidade, os conceitos relacionados à UML.

SOFTWARE

De acordo com Sommerville (2007, p. 4), o software é composto não somente pelos programas, mas também pela documentação associada a esses programas.

Para Pressman (2011, p. 32), o software possui, pelo menos, três características que o diferenciam do hardware:

1. Software é desenvolvido ou passa por um processo de engenharia, não sendo fabricado no sentido clássico.

   Apesar de existir semelhanças entre o desenvolvimento de software e a fabricação de hardware, essas atividades são muito diferentes. Os custos de software concentram-se no processo de engenharia, por causa disso os projetos de software não podem ser conduzidos como se fossem projetos de fabricação.

2. Software não se desgasta.

   Normalmente, o hardware apresenta taxas de defeitos mais altas no início de sua vida, porém esses defeitos são corrigidos tendo assim a taxa decrescida, ou seja, atingindo um nível estável. Porém, com o uso, o hardware pode se desgastar devido à poeira, má utilização, temperaturas extremas e outros. Já com o software é diferente, ou seja, ele não está sujeito aos problemas ambientais, como o hardware. Em relação aos problemas iniciais, com o software também acontece assim, pois alguns defeitos ainda podem passar pela etapa de validação do software.

Quando um componente de hardware se desgasta, ele normalmente é trocado por um novo componente e o hardware volta a funcionar. Com o software isso não acontece, não tem como simplesmente trocar o componente, pois quando um erro acontece no hardware, pode ser de projeto, de requisito mal definido, levando o software a sofrer manutenção, que pode ser considerada mais complexa que a do hardware.

Embora a indústria caminhe para a construção com base em componentes, a maioria

Anotações

dos softwares continua a ser construída de forma personalizada (sob encomenda).

A reutilização de componentes de hardware é parte natural do seu processo de engenharia, já para o software é algo que apenas começou a ser alcançado (PRESSMAN, 2011, p. 34). Os componentes reutilizáveis de software são criados para que o desenvolvedor possa se concentrar nas partes do projeto que representam algo novo. Esses componentes encapsulam dados e o processamento aplicado a eles, possibilitando criar novas aplicações a partir de partes reutilizáveis. Na unidade II trataremos sobre Engenharia de Software Orientada a Reuso.

ENGENHARIA DE SOFTWARE

De acordo com Sommerville (2011, p. 5), “a engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção”.

Como dissemos na introdução desta unidade, o termo engenharia de software é relativamente novo e foi proposto para tornar o desenvolvimento de software sistemático, podendo ser realizado com padrões de qualidade, dentro do cronograma e do orçamento previstos inicialmente.

Para Sommerville (2011, p. 5), a engenharia de software é importante por dois motivos:

1. A sociedade, cada vez mais, depende de sistemas de software avançados, portanto é preciso que os engenheiros de software sejam capazes de produzir sistemas confiáveis de maneira econômica e rápida.
2. A longo prazo, normalmente, é mais barato usar métodos e técnicas propostos pela engenharia de software, ao invés de somente escrever os programas como se fossem algum projeto pessoal. Para a maioria dos sistemas, o maior custo está na sua manutenção, ou seja, nas alterações que são realizadas depois que o sistema é implantado.

Anotações

Alguns Fundamentos da Engenharia de Software

Por Wilson de Pádua Paula Filho

O que é Engenharia de Software?

É a mesma coisa que Ciência da Computação? Ou é uma entre muitas especialidades da Ciência da Computação? Ou dos Sistemas de Informação, ou do Processamento de Dados, ou da Informática, ou da Tecnologia da Informação? Ou é uma especialidade diferente de todas as anteriores?

Na maioria das instituições brasileiras de ensino superior, o conjunto de conhecimentos e técnicas conhecido como Engenharia de Software é ensinado em uma ou duas disciplinas dos cursos que têm os nomes de Ciência da Computação, Informática ou Sistemas de Informação. Raramente, em mais disciplinas, muitas vezes opcionais, e muitas vezes oferecidas apenas em nível de pós-graduação. Algumas instituições oferecem cursos de pós-graduação em Engenharia de Software, geralmente no nível de especialização.

Neste artigo você vai entender os fundamentos da engenharia de software. O artigo completo pode ser acessado no endereço abaixo.

Fonte: <http://www.devmedia.com.br/artigo-engenharia-de-software-alguns-fundamentos-da-engenharia-de-software/8029>. Acesso em: 02 jun. 2012.

TIPOS DE APLICAÇÕES DE SOFTWARE

Atualmente, com o software sendo utilizado em praticamente todas as atividades exercidas pelas pessoas, existe uma grande variedade de aplicações de software. Vamos ver algumas delas:

- **Software de sistema** – de acordo com Pressman (2011, p. 34), são aqueles programas desenvolvidos para atender a outros programas, como por exemplo, editores de texto, compiladores e sistemas operacionais.
- **Software de aplicação** – são programas desenvolvidos para solucionar uma neces-

Anotações

sidade específica de negócio, processando dados comerciais ou técnicos de forma que ajude nas operações comerciais ou tomadas de decisão pelos administradores da empresa.

- **Software científico/de engenharia** – são aplicações que vão da astronomia à vulcanologia, da biologia molecular à fabricação automatizada, normalmente utilizando algoritmos para processamento numérico pesado.
- **Software embutido** – controla ou gerencia dispositivos de hardware, como por exemplo, celular, painel do micro-ondas, controle do sistema de freios de um veículo.
- **Software de inteligência artificial** – utiliza algoritmos não numéricos para solucionar problemas complexos que não poderiam ser solucionados pela computação ou análise direta, como por exemplo, sistemas especialistas, robótica, redes neurais artificiais.

"Não existe computador que tenha bom senso".

Marvin Minsky, cofundador do laboratório de inteligência artificial do Instituto de Tecnologia de Massachusetts.

FERRAMENTAS CASE (*COMPUTER-AIDED SOFTWARE ENGINEERING*)

Uma ferramenta CASE é um software que pode ser utilizado para apoiar as atividades do processo de software, como a engenharia de requisitos, o projeto, o desenvolvimento de programa e os testes. As ferramentas CASE podem incluir editores de projeto, dicionários de dados, compiladores, depuradores, ferramentas de construção de sistemas entre outros (SOMMERVILLE, 2007, p. 56).

Sommerville (2007, p. 56) mostra alguns exemplos de atividades que podem ser automatizadas utilizando-se CASE, entre elas:

1. O desenvolvimento de modelos gráficos de sistemas, como parte das especificações de requisitos ou do projeto de software.
2. A compreensão de um projeto utilizando-se um dicionário de dados que contém informações sobre as entidades e sua relação em um projeto.

3. A geração de interfaces com usuários, a partir de uma descrição gráfica da interface, que é criada interativamente pelo usuário.
4. A depuração de programas, pelo fornecimento de informações sobre um programa em execução.
5. A tradução automatizada de programas, a partir de uma antiga versão de uma linguagem de programação, como Cobol, para uma versão mais recente.

A tecnologia CASE está atualmente disponível para a maioria das atividades de rotina no processo de software, proporcionando assim algumas melhorias na qualidade e na produtividade de software.

Existe, basicamente, duas formas de classificação geral para as Ferramentas CASE. A primeira delas as divide em *Upper* CASE, *Lower* CASE, *Integrated*-CASE e *Best in Class*:

- *Upper* CASE ou U-CASE ou *Front-End*: são aquelas ferramentas que estão voltadas para as primeiras fases do processo de desenvolvimento de sistemas, como análise de requisitos, projeto lógico e documentação. São utilizadas por analistas e pessoas mais interessadas na solução do problema do que na implementação.
- *Lower* CASE ou L-CASE ou *Back-End*: é praticamente o oposto das anteriormente citadas, oferecem suporte nas últimas fases do desenvolvimento, como codificação, testes e manutenção.
- *Integrated* CASE ou I-CASE: este tipo de ferramenta, além de englobar características das *Upper* e *Lower* CASE's, ainda oferecem outras características, como por exemplo, controle de versão. Entretanto, esse tipo de ferramenta somente é utilizado em projetos de desenvolvimento muito grandes, por serem bastante caras e difíceis de operar.
- *Best in Class* ou Kit de Ferramenta: semelhante as I-CASE, os Kits de Ferramenta também acompanham todo o ciclo de desenvolvimento, entretanto, possuem a propriedade de conjugar sua capacidade a outras ferramentas externas complementares, que variam de acordo com as necessidades do usuário. Para um melhor entendimento, podemos compará-las a um computador sem o kit multimídia, as *Best in Class* seriam o computador que funciona normalmente, e as outras ferramentas fariam o papel do kit multimídia, promovendo um maior potencial de trabalho à má-

Anotações

quina. São mais usadas para desenvolvimento corporativo, apresentando controle de acesso, versão, repositórios de dados entre outras características.

A segunda forma divide as Ferramentas CASE em orientadas à função e orientadas à atividade:

- Orientadas à função: seriam as *Upper* CASE e *Lower* CASE. Baseiam-se na funcionalidade das ferramentas, ou seja, são aquelas que têm funções diferentes no ciclo de vida de um projeto, como representar apenas o Diagrama de Entidades e Relacionamentos (DER) ou o Diagrama de Fluxo de Dados (DFD).
- Orientadas a atividade: Seriam as *Best In Class* e as I-CASE, que processam a atividades como especificações, modelagem e implementação.

CONCEITOS BÁSICOS DE ORIENTAÇÃO A OBJETOS

A UML é totalmente baseada no paradigma de orientação a objetos (OO). Sendo assim, para compreendê-la corretamente, precisamos antes estudar alguns dos conceitos relacionados à orientação a objetos.

Objetos

Segundo Melo (2004, p.15), um objeto é qualquer coisa, em forma concreta ou abstrata, que

Anotações

exista física ou apenas conceitualmente no mundo real. São exemplos de objetos: cliente, professor, carteira, caneta, carro, disciplina, curso, caixa de diálogo. Os objetos possuem características e comportamentos.

Classes

Uma classe é uma abstração de um conjunto de objetos que possuem os mesmos tipos de características e comportamentos, sendo representada por um retângulo que pode possuir até três divisões. A primeira divisão armazena o nome da classe; a segunda, os atributos (características) da classe; e a terceira, os métodos. Geralmente, uma classe possui atributos e métodos, porém, é possível encontrar classes que contenham apenas uma dessas características ou mesmo nenhuma quando se tratar de classes abstratas. A figura abaixo apresenta um exemplo de classe.

Exemplo de uma Classe

De acordo com Melo (2004, p. 18), o ponto inicial para identificação de classes num documento de requisitos é assinalar os substantivos. Note que esses substantivos podem levar a objetos físicos (como cliente, livro, computador) ou objetos conceituais (como reserva, cronograma, norma). Em seguida, é preciso identificar somente os objetos que possuem importância para o sistema, verificando o que realmente consiste em objeto e o que consiste em atributo. Ainda é preciso fazer uma nova análise dos requisitos para identificar classes implícitas no contexto trabalhado, excluir classes parecidas ou juntar duas classes em uma única classe. Vale a pena ressaltar que esse processo será iterativo e não será possível definir todas as classes de uma só vez.

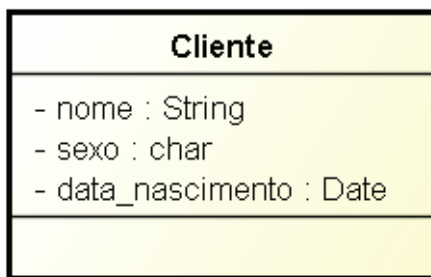
Anotações

Atributos

Uma classe, normalmente, possui atributos que representam as características de uma classe, que costumam variar de objeto para objeto, como o nome em um objeto da classe Cliente ou a cor em um objeto da classe Carro, e que permitem diferenciar um objeto de outro da mesma classe.

De acordo com Guedes (2007, p. 32), na segunda divisão da classe aparecem os atributos, sendo que cada atributo deve possuir um nome e, eventualmente, o tipo de dado que o atributo armazena, como, por exemplo, integer, float ou char. Assim, a classe especifica a estrutura de um objeto sem informar quais serão os seus valores, sendo que um objeto corresponde a uma instância de uma classe em um determinado momento. Vou mostrar um exemplo para facilitar o entendimento:

Uma classe pessoa possui os atributos nome, sexo e data de nascimento. Essa classe pode ter um objeto ou instância com os seguintes valores para cada atributo, respectivamente, Márcia, feminino e 22/03/1969. Dessa forma, em um sistema, devemos trabalhar com as instâncias (objetos) de uma classe, pois os valores de cada atributo são carregados nas instâncias (MELO, 2004, p. 17). A figura abaixo mostra um exemplo de classe com atributos.

Cliente
- nome : String - sexo : char - data_nascimento : Date

Exemplo de Classe com Atributos

Anotações

Métodos

Métodos são processos ou serviços realizados por uma classe e disponibilizados para uso de outras classes em um sistema e devem ficar armazenados na terceira divisão da classe. Os métodos são as atividades que uma instância de uma classe pode executar (GUEDES, 2007, p. 33).

Um método pode receber ou não parâmetros (valores que são utilizados durante a execução do método) e pode, também, retornar valores, que podem representar o resultado da operação executada ou somente indicar se o processo foi concluído com sucesso ou não. Assim, um método representa um conjunto de instruções que são executadas quando o método é chamado. Um objeto da classe Cliente, por exemplo, pode executar a atividade de incluir novo cliente. A figura a seguir apresenta um exemplo de uma classe com métodos.

Cliente
- nome : String - sexo : char - data_nascimento : Date
+ IncluirNovoCliente() : void + AtualizarCliente() : void

Exemplo de Classe com Atributos e Métodos

Visibilidade

De acordo com Guedes (2007, p. 34), a visibilidade é um símbolo que antecede um atributo ou método e é utilizada para indicar seu nível de acessibilidade. Existem basicamente três modos de visibilidade: público, protegido e privado.

- O símbolo de mais (+) indica visibilidade pública, ou seja, significa que o atributo ou método pode ser utilizado por objetos de qualquer classe.

Anotações

- O símbolo de sustenido (#) indica que a visibilidade é protegida, ou seja, determina que apenas objetos da classe possuidora do atributo ou método ou de suas subclasses podem acessá-lo.
- O símbolo de menos (-) indica que a visibilidade é privada, ou seja, somente os objetos da classe possuidora do atributo ou método poderão utilizá-lo.

Note que no exemplo da classe Cliente, tanto os atributos quanto os métodos apresentam sua visibilidade representada à esquerda de seus nomes, em que os atributos nome, sexo e data_nascimento possuem visibilidade privada, pois apresentam o símbolo de menos (-) à esquerda da sua descrição. Já os métodos IncluirNovoCliente() e AtualizarCliente(), possuem visibilidade pública, como indica o símbolo + acrescentado à esquerda da sua descrição.

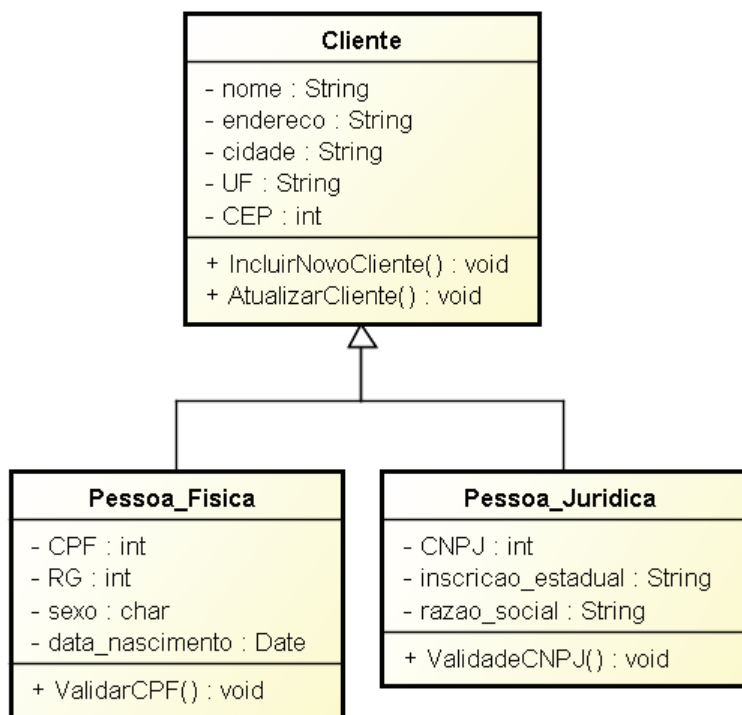
Herança (Generalização/Especialização)

A herança permite que as classes de um sistema compartilhem atributos e métodos, otimizando assim o tempo de desenvolvimento, com a diminuição de linhas de código, facilitando futuras manutenções (GUEDES, 2007, p. 35). A herança (ou generalização/especialização) acontece entre classes gerais (chamadas de superclasses ou classes-mãe) e classes específicas (chamadas de subclasses ou classes-filha) (BOOCH, 2005, p. 66).

A herança significa que os objetos da subclasse podem ser utilizados em qualquer local em que a superclasse ocorra, mas não vice-versa. A subclasse herda as propriedades da mãe, ou seja, seus atributos e métodos, e também podem possuir atributos e métodos próprios, além daqueles herdados da classe-mãe.

De acordo com Guedes (2007, p.35), a vantagem do uso da herança é que, quando uma classe é declarada com seus atributos e métodos específicos e após isso uma subclasse é derivada, não é necessário redeclarar os atributos e métodos já definidos, ou seja, por meio da herança a subclasse os herda automaticamente, permitindo a reutilização do código já pronto. Assim, é preciso somente declarar os atributos ou métodos restritos da subclasse, tornando o processo de desenvolvimento mais ágil, além de facilitar as manutenções futuras, pois, em caso de

Anotações

uma alteração, será necessário somente alterar o método da superclasse para que todas as subclasses estejam também atualizadas. A figura abaixo apresenta um exemplo de herança, explicitando os papéis de superclasse e subclasse e apresentando também o símbolo de herança da UML, uma linha que liga as classes com um triângulo tocando a superclasse.

Exemplo de Herança (generalização/especialização)

Fonte: a autora

A figura acima mostra a superclasse Cliente com os atributos *nome, endereço, cidade, UF e CEP* e os métodos *IncluirNovoCliente() e AtualizarCliente().* A subclasse Pessoa_Fisica herda esses atributos e métodos, além de possuir os atributos *CPF, RG, sexo e data_nascimento* e o método *ValidarCPF()*. A seta que aponta para a superclasse Cliente indica a herança.

Anotações

A subclasse Pessoa_Juridica também herda todos os atributos e métodos da superclasse Cliente, além de possuir os atributos *CNPJ, inscrição_estadual e razão_social* e o método *ValidarCNPF()*.

Quando olhamos a figura acima, notamos que a classe Cliente é a mais genérica e as classes Pessoa_Fisica e Pessoa_Juridica são as mais especializadas. Então, podemos dizer que generalizamos quando partimos das subclasses para a superclasse, e especializamos quando fazemos o contrário, ou seja, quando partimos superclasse para as subclasses.

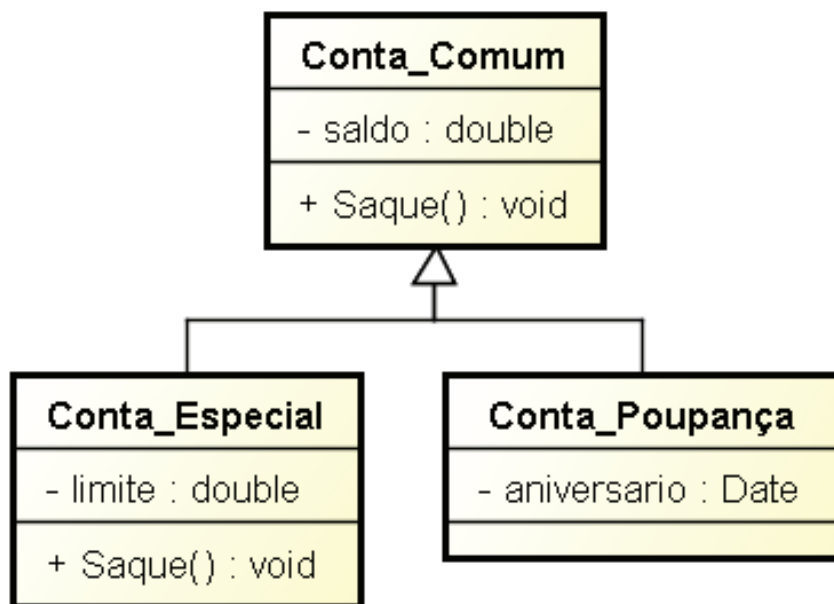
Polimorfismo

O conceito de polimorfismo está associado à herança, pois o mesmo trabalha com a redeclaração de métodos previamente herdados por uma classe. Esses métodos, embora parecidos, diferem de alguma forma da implementação utilizada na superclasse, sendo preciso implementá-los novamente na subclasse. Mas, a fim de não ter que alterar o código-fonte, acrescentando uma chamada a um método com um nome diferente, redeclara-se o método com o mesmo nome declarado na superclasse (GUEDES, 2007, p. 36).

De acordo com Lima (2009, p. 26), polimorfismo é o princípio em que classes derivadas (as subclasses) e uma mesma superclasse podem chamar métodos que têm o mesmo nome (ou a mesma assinatura), mas possuem comportamentos diferentes em cada subclasse, produzindo resultados diferentes, dependendo de como cada objeto implementa o método.

Ou seja, podem existir dois ou mais métodos com a mesma nomenclatura, diferenciando-se na maneira como foram implementados, sendo o sistema responsável por verificar se a classe da instância em questão possui o método declarado nela própria ou se o herda de uma superclasse (GUEDES, 2007, p. 36).

Por ser um exemplo bastante claro, para ilustrar o polimorfismo, utilizaremos o mesmo exemplo de Gedues (2007, p. 37), que está mostrado na figura abaixo.

Anotações

Exemplo de Polimorfismo

Fonte: Guedes (2007, p. 37)

No exemplo apresentado acima, aparece uma classe geral chamada Conta_Comum (que, nesse caso, é a superclasse), que possui um atributo chamado Saldo, contendo o valor total depositado em uma determinada instância da classe e um método chamado Saque. Esse método somente diminui o valor a ser debitado do saldo da conta se este possuir o valor suficiente. Caso contrário, a operação não poderá ser realizada, ou seja, o saque deve ser recusado pelo sistema (GUEDES, 2007, p. 37).

A partir da superclasse Conta_Comum, uma nova classe foi derivada, a subclasse Conta_Especial, que possui um atributo próprio chamado limite e possui também os atributos herdados da superclasse. O atributo limite define o valor extra que pode ser sacado além do valor contido no saldo da conta. Por esse motivo, a classe Conta_Especial apresenta uma

Anotações

redefinição do método Saque, porque a rotina do método Saque da classe Conta_Especial é diferente a do método Saque declarado na classe Conta_Comum, pois é preciso incluir o limite da conta no teste para determinar se o cliente pode ou não retirar o valor solicitado. No entanto, o nome do método permanece o mesmo; apenas no momento de executar o método, o sistema deverá verificar se a instância que o chamou pertence à classe Conta_Comum ou à classe Conta_Especial, executando o método definido na classe em questão (GUEDES, 2007, p. 37).

<http://www.youtube.com/watch?v=MnJLeYAno4o&feature=relmfu>.
Vídeo que mostra uma introdução aos conceitos de orientação a objetos.

UML – *UNIFIED MODELING LANGUAGE*

Fonte: <http://onlywhatmatters.wordpress.com/2011/02/20/uml-unified-modeling-language/>

Anotações

Segundo Booch (2005, p. 13), "a UML (*Unified Modeling Language* ou Linguagem de Modelagem Unificada) é uma linguagem-padrão para a elaboração da estrutura de projetos de software", podendo ser utilizada para a visualização, especificação, construção e documentação de artefatos de software, por meio do paradigma de Orientação a Objetos. A UML tem sido a linguagem-padrão de modelagem de software adotada internacionalmente pela indústria de Engenharia de Software (GUEDES, 2007, p. 13).

A UML não é uma linguagem de programação, mas uma linguagem de modelagem, que tem como meta auxiliar os engenheiros de software a definir as características do software, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado. Todas essas características são definidas por meio da UML antes do início do desenvolvimento do software (GUEDES, 2007, p. 13).

De acordo com Booch (2005, p. 13), a UML é apenas uma linguagem de modelagem e é independente de processo de software, podendo ser utilizada em modelo cascata, desenvolvimento evolucionário, ou qualquer outro processo que esteja sendo utilizado para o desenvolvimento do software.

A notação UML utiliza diversos símbolos gráficos, existindo uma semântica bem definida para cada um deles, sendo possível elaborar diversos modelos. A UML tem sido empregada de maneira efetiva em sistemas cujos domínios abrangem: sistemas de informações corporativos, serviços bancários e financeiros, transportes, serviços distribuídos baseados na Web entre outros. Porém, a UML não se limita à modelagem de software, podendo modelar sistemas como o fluxo de trabalho no sistema legal, a estrutura e o comportamento de sistemas de saúde e o projeto de hardware (BOOCH, 2005, p. 17).

Anotações

HISTÓRICO DA UML

A UML originou-se da junção dos Método Booch de Grady Booch, Método OMT (*Object Modeling Technique*) de Rumbaugh e do método OOSE (*Object-Oriented Software Engineering*) de Jacobson. Esses eram, até meados da década de 1990, as três metodologias de modelagem orientadas a objetos mais populares entre os profissionais da área de engenharia de software (GUEDES, 2007, p. 13).

Em outubro de 1994, Rumbaugh se juntou a Booch na *Rational Software Corporation*, iniciando assim, oficialmente, os esforços para a criação da UML. O foco inicial do projeto era a unificação dos métodos Booch e OMT, o que resultou no lançamento do Método Unificado no final de 1995. Logo em seguida, Jacobson juntou-se a Booch e Rumbaugh na *Rational Software* e seu método OOSE começou também a ser incorporado à nova metodologia (BOOCH, 2005). O trabalho de Booch, Jacobson e Rumbaugh, conhecidos popularmente como "Os Três Amigos", resultou no lançamento, em 1996, da primeira versão da UML propriamente dita, que foi chamada de versão 0.9.

Tão logo a primeira versão foi lançada, diversas empresas de software passaram a contribuir com o projeto, estabelecendo um consórcio de UML com várias empresas que desejavam dedicar recursos com o objetivo de trabalhar para uma definição mais forte e completa da UML, criando-se assim a versão 1.0 da UML. Essa versão foi oferecida para padronização ao OMG (*Object Management Group* ou Grupo de Gerenciamento de Objetos) em janeiro de 1997.

De acordo com Booch (2005), entre janeiro e julho de 1997, o grupo original de parceiros cresceu e passou a incluir praticamente todos os participantes e colaboradores da resposta inicial ao OMG, criando uma versão revisada da UML (versão 1.1), que foi novamente oferecida para padronização ao OMG.

Finalmente, a UML foi adotada pela OMG em novembro de 1997, como uma linguagem padrão

Anotações

de modelagem, sendo que sua manutenção ficou sob responsabilidade da RTF (*Revision Task Force*), pertencente à OMG. O objetivo da RTF é realizar revisões nas especificações, referentes a erros, inconsistências, ambiguidades e pequenas omissões, de acordo com os comentários da comunidade em geral (MELO, 2004, p. 31). Porém, essas revisões não devem provocar uma grande mudança no escopo original da proposta de padronização. Nestes últimos anos aconteceram as seguintes revisões: em julho de 1998, a UML 1.2; no final de 1998, a UML 1.3; em maio de 2001, a UML 1.4. Em agosto de 2001, a RTF submeteu ao OMG um relatório provisório da UML 1.5, publicada em março de 2003. No início de 2005, a versão oficial da UML 2.0, foi adotado pelo OMG. Hoje, a UML está em sua versão 2.4.1 e sua documentação oficial pode ser consultada através do endereço <www.uml.org>. A grande mudança aconteceu na versão 2.0, sendo que a maioria da bibliografia disponível atualmente, inclusive a que está sendo utilizada na consulta para produção deste livro, trata dessa versão.

FERRAMENTAS CASE BASEADAS NA LINGUAGEM UML

Nesta unidade, **já vimos que uma ferramentas CASE (***Computer-Aided Software Engineering* – Engenharia de Software Auxiliada por Computador) é um software que, de alguma forma, colabora na realização de uma ou mais atividades realizadas durante o processo de desenvolvimento de software. Agora vamos ver alguns exemplos de ferramentas CASE que suportam a UML, sendo esta, em geral, sua principal característica. Existem diversas ferramentas no mercado, dentre as quais, podemos destacar:

Rational Rose: a ferramenta *Rational Rose* foi desenvolvida pela *Rational Software Corporation*, empresa que estimulou a criação da UML, sendo a primeira ferramenta CASE baseada na linguagem UML. Atualmente, essa ferramenta é bastante usada pelas empresas desenvolvedoras de software. Ela permite a modelagem dos diversos diagramas da UML e também a construção de modelos de dados com possibilidade de exportação para construção da base de dados ou realização de engenharia reversa de uma base de dados existente. Em

Anotações

____

____

____

____

20 de fevereiro de 2003, a empresa *Rational* foi adquirida pela IBM e agora a ferramenta chama-se sucedido pelo IBM *Rational Architect*. Maiores informações sobre esta ferramenta podem ser consultadas no endereço <www.rational.com>.

Astah Professional: é uma ferramenta para criação de diagramas UML, possuindo uma versão gratuita, o *Astah Community*, e outras versões pagas. A versão gratuita, que pode ser obtida no endereço <http://astah.net/download>, possui algumas restrições de funções, porém para as que precisaremos nesta unidade será suficiente, portanto, será essa a ferramenta que utilizaremos para modelar os diagramas da UML que aprenderemos. Anteriormente, essa ferramenta era conhecida por *Jude*.

***Visual Paradigm for UML* ou VP-UML**: esta ferramenta oferece uma versão que pode ser baixada gratuitamente, a *Community Edition*, porém essa versão não suporta todos os serviços e opções disponíveis nas versões *Standard* ou *Professional* da ferramenta. Mas, para quem deseja praticar a UML, a versão gratuita é uma boa alternativa, apresentando um ambiente amigável e de fácil compreensão. O download das diferentes versões da ferramenta pode ser feito em: <http://www.visual-paradigm.com>.

Enterprise Architect: esta ferramenta não possui uma edição gratuita como as anteriores, porém é uma das ferramentas que mais oferecem recursos compatíveis com a UML 2. Uma versão Trial para avaliação dessa ferramenta pode ser obtida através do site <www. sparxsystems.com.au>.

CONSIDERAÇÕES FINAIS

Nesta primeira unidade foram apresentados alguns conceitos básicos sobre engenharia de software que serão utilizados no decorrer de todo o livro, por isso, é muito importante que esses conceitos fiquem bem claros para você.

A engenharia de software foi proposta para tentar levar a precisão da engenharia para o desenvolvimento de software, pois até aquela época, desenvolver um software era algo que não podia ser mensurado, nem em relação ao tempo, nem em relação ao custo, levando-se, normalmente, muito mais tempo do que o previsto. E o que acontecia era que não se tinha uma regra, uma sequência de atividades para o desenvolvimento. Você vai ver na próxima unidade que, para tentar solucionar esse problema, os estudiosos da engenharia de software

propuseram vários modelos de processos de software, sendo que a empresa pode escolher o que melhor se adequa a ela. Isso tem ajudado muito o desenvolvimento de software. Você vai perceber isso durante o estudo das próximas unidades.

Outro conceito importante que foi tratado nesta primeira unidade é o conceito de software. Algumas pessoas que conheço acham que desenvolver software é simplemesmente sentar em frente ao computador e escrever as linhas de código. Você percebeu que sim, isso faz parte do software, mas que desenvolver software é muito mais abrangente, pois o software envolve, além dos programas, a documentação, as pessoas, os procedimentos envolvidos. A engenharia de software trata de todos esses aspectos, mas em nosso livro trataremos mais especificamente da modelagem de um software, desde o levantamento dos requisitos até a elaboração de vários diagramas. Não trataremos da implementação propriamente dita, pois isso você verá em outras disciplinas do curso. A implementação é muito importante, por isso você terá disciplinas dedicadas a essa tarefa.

Todas as etapas da engenharia de software podem ser desenvolvidas com o auxílio de ferramentas CASE, que, conforme vimos, nada mais é do que um software desenvolvido por alguma empresa, e que tem por objetivo auxiliar o desenvolvedor nas tarefas executadas durante o desenvolvimento (desde o seu início até a implantação do software para o usuário). Em nossa disciplina de engenharia de software vamos utilizar a ferramenta *Astah*, que você pode baixar gratuitamente no endereço mencionado nesta unidade. Sua instalação é bastante simples e você já pode deixar a ferramenta instalada para uso nas próximas unidades.

Estudamos também vários conceitos de orientação a objetos que utilizaremos em nossa disciplina e, com certeza, você também vai utilizar quando for estudar, por exemplo, alguma linguagem orientada ao objeto, como Java. Portanto, o entendimento desses conceitos será de suma importância para todo o curso.

E, finalmente, apresentei a você um breve histórico do que é a UML e como ela foi concebida. Utilizaremos a linguagem UML para a elaboração de diversos diagramas. Note que essa é uma linguagem padrão, universal, ou seja, um diagrama produzido em nossa disciplina pode ser lido por alguém que conheça UML e que esteja do outro lado do mundo.

Essa primeira unidade foi somente um aperitivo. Agora vamos passar a estudar assuntos específicos da disciplina e você vai perceber que a engenharia de software é muito importante

para o desenvolvimento de um software.

ATIVIDADE DE AUTOESTUDO

1. Uma classe é um conjunto de objetos que compartilham os mesmos atributos, métodos e relacionamentos. Sendo assim:
    a) Explique o que significa instanciar uma classe.
    b) Descreva o que são atributos.
2. Um dos principais recursos da orientação a objetos é a Herança entre classes, uma vez que esse recurso pode reduzir substancialmente as repetições nos projetos e programas orientados a objetos. Dentro deste contexto:

a) Explique o que é Herança.

b) Crie um exemplo de Herança com no mínimo três classes. Neste exemplo devem aparecer atributos tanto na superclasse quanto nas subclasses.

UNIDADE II

PROCESSOS DE SOFTWARE

Professora Me. Márcia Cristina Dadalto Pascutti

Objetivos de Aprendizagem

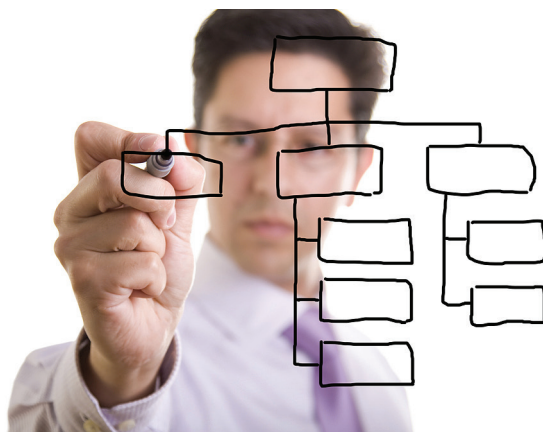
- Compreender os conceitos de processo de software e modelos de processo de software.
- Mostrar as atividades básicas do processo de software.
- Mostrar três modelos de processo de software.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- **Processos de Software**
- **Modelos de Processo de Software**
- **Atividades Básicas do Processo de Software**

INTRODUÇÃO

Caro(a) aluno(a), na primeira unidade você aprendeu alguns conceitos básicos relacionados à Engenharia de Software. A partir de agora você começará a estudar assuntos mais específicos da disciplina e você verá que seu estudo ficará cada vez mais interessante.

Nos últimos tempos, o desenvolvimento de software é uma das áreas da tecnologia que mais cresceu, e com esse crescimento vieram também problemas que vão desde o não cumprimento dos prazos estabelecidos para o seu desenvolvimento até a falta de qualidade do software desenvolvido.

Um software não pode ser desenvolvido de qualquer jeito, sem seguir critérios, sem que se saiba qual o próximo passo a ser dado. Por isso que os conceitos relacionados à engenharia de software devem ser utilizados. Hoje em dia, a empresa precisa definir qual o seu processo de software.

Nesta unidade, você aprenderá o que é um processo de software e conhecerá alguns modelos de processo que já existem em nossa literatura e que são utilizados por muitas empresas. Esses modelos são: modelo em cascata, desenvolvimento incremental e engenharia de software baseada em componentes. Mas, é importante ressaltar que não é preciso seguir um

Anotações

desses modelos que já estão prontos, ou seja, a empresa que vai desenvolver software pode criar o seu próprio modelo. É imprescindível que esse modelo seja seguido.

Existem algumas atividades básicas que fazem parte de um processo de software. Estudaremos cada uma dessas atividades: especificação de software, projeto e implementação de software, validação de software e evolução de software. Você perceberá que os modelos de processo de software apresentados nesta unidade possuem todas essas atividades, e que, às vezes, a atividade possui um nome diferente, mas com o mesmo significado. Você verá também que uma atividade pode se desdobrar em várias etapas ou subatividades.

PROCESSOS DE SOFTWARE

Para que um software seja produzido são necessárias diversas etapas, compostas de uma série de tarefas em cada uma delas. A esse conjunto de etapas dá-se o nome de processo de software. Essas etapas podem envolver o desenvolvimento de software a partir do zero em uma determinada linguagem de programação (por exemplo, o Java ou C) ou então a ampliação e a modificação de sistemas já em utilização pelos usuários.

Segundo Sommerville (2011), existem muitos processos de software diferentes, porém todos devem incluir quatro atividades fundamentais para a engenharia de software:

1. **Especificação de software.** É necessário que o cliente defina as funcionalidades do software que será desenvolvido, bem como defina todas as suas restrições operacionais.
2. **Projeto e implementação de software.** O software deve ser confeccionado seguindo as especificações definidas anteriormente.
3. **Validação de software**. O software precisa ser validado para garantir que ele faz o que o cliente deseja, ou seja, que ele atenda às especificações de funcionalidade.
4. **Evolução de software.** As funcionalidades definidas pelo cliente durante o desenvolvimento do software podem mudar e o software precisa evoluir para atender a essas mudanças.

Anotações

Vamos estudar detalhadamente cada uma das atividades mencionadas acima durante a nossa disciplina, inclusive utilizando ferramentas automatizadas (ferramentas CASE, já estudadas em nossa unidade I) para nos auxiliar na elaboração dos diversos documentos que serão necessários.

Para Sommerville (2011, p. 19), “os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software”.

Mas o que acontece é que nem sempre as empresas aproveitam as boas técnicas da engenharia de software em seu desenvolvimento de software. E, normalmente, o software não atende aos requisitos do usuário, acaba demorando mais tempo para ser desenvolvido do que o previsto, aumentando assim, o valor do custo do software.

As atividades mencionadas por Sommerville podem ser organizadas pelas empresas da forma como ela achar melhor, porém, é importante ressaltar que todas essas atividades são de extrema importância e o processo de software adotado pela empresa não deve deixar de considerar nenhuma das etapas.

O processo de desenvolvimento de software e a utilização de ferramentas CASE

Por Maria Clara dos Santos Pinto Silveira

A presente dissertação situa-se na área das ferramentas CASE. É apresentado um estudo dos conceitos fundamentais da Engenharia de Software e são abordados diversos problemas relacionados com o desenvolvimento de software.

São também apresentados os paradigmas mais conhecidos e algumas metodologias para o desenvolvimento de software. Registra ainda as características, vantagens e desvantagens das ferramentas

Anotações

CASE.

Nesta Dissertação é efetuado um estudo sobre a sistematização do caminho a percorrer na escolha de um ambiente CASE. Para tal são analisadas questões como: metodologia a utilizar, decisão a tomar quanto ao produto ou produtos que correspondem às necessidades e capacidades da organização, seleção do fornecedor, nível de formação exigida e custos envolvidos.

Para ilustrar este estudo foi desenvolvida uma aplicação que permite ao departamento de qualidade de uma indústria de lacticínios gerir todas as amostras e análises efetuadas ao nível do produtor e do processo de fabrico. Nesta aplicação foram usadas ferramentas CASE, EasyCASE Professional 4.22 e EasyCASE Database Engineer 1.10, assim como uma base de dados, Microsoft Access 2.0.

Fonte: <http://repositorio-aberto.up.pt/handle/10216/12914>. Acesso em: 02 jun. 2012.

É importante ressaltar que mesmo as empresas que possuem um processo de software bem definido e documentado, para determinados softwares que ela desenvolve, pode ser utilizado outro processo de software, ou seja, dependendo do software a ser desenvolvido, pode ser utilizado um determinado processo de software. Na próxima seção veremos alguns modelos de processo de software e veremos também que é possível a empresa adotar um processo de software próprio, que atenda as suas necessidades.

MODELOS DE PROCESSO DE SOFTWARE

Como foi discutido anteriormente, um processo de software é composto por um conjunto de etapas que são necessárias para que um software seja produzido. Sommerville (2007) diz que um modelo de processo de software é uma representação abstrata, simplificada, de um processo de software. Os modelos de processo incluem as atividades que fazem parte do processo de software (você está lembrado das atividades descritas no item anterior?), os artefatos de software que devem ser produzidos em cada uma das atividades (documentos) e também os papéis das pessoas envolvidas na engenharia de software. Além disso, cada modelo de processo representa um processo a partir de uma perspectiva particular, de uma maneira que proporciona apenas informações parciais sobre o processo.

Na literatura existem diversos modelos de processo de software. Aqui irei mostrar somente três desses modelos e, em seguida, mostrarei as atividades básicas que estão presentes em, praticamente, todos os modelos de processos de software.

Os modelos de processo que mostrarei foram retirados de Sommerville (2011, p.20) e são:

1. **Modelo em Cascata**. Esse modelo considera as atividades de especificação, desenvolvimento, validação e evolução, que são fundamentais ao processo, e as representa como fases separadas, como a especificação de requisitos, o projeto de software, a implementação, os testes e assim por diante (SOMMERVILLE, 2011).

2. **Desenvolvimento Incremental.** Esse modelo intercala as atividades de especificação, desenvolvimento e validação. Um sistema inicial é rapidamente desenvolvido a partir de especificações abstratas, que são então refinadas com informações do cliente, para produzir um sistema que satisfaça a suas necessidades, produzindo várias versões do software (SOMMERVILLE, 2011).

3. **Engenharia de Software Orientada a Reuso.** Esse modelo parte do princípio de que existem muitos componentes que podem ser reutilizáveis. O processo de desenvolvimento do sistema se concentra em combinar vários desses componentes em um sistema, em vez de proceder ao desenvolvimento a partir do zero, com o objetivo de reduzir o tempo de desenvolvimento (SOMMERVILLE, 2011).

O modelo em cascata

Anotações

O modelo cascata ou ciclo de vida clássico, considerado o paradigma mais antigo da engenharia de software, sugere uma abordagem sequencial e sistemática para o desenvolvimento de software, começando com a definição dos requisitos por parte do cliente, avançando pelas atividades de projeto e implementação de software, testes, implantação, culminando no suporte contínuo do software concluído.

Segundo Sommerville (2007, p. 44), os principais estágios do modelo em cascata demonstram as atividades fundamentais do desenvolvimento:

1. **Análise e definição de requisitos** – As funções, as restrições e os objetivos do sistema são estabelecidos por meio da consulta aos usuários do sistema. Em seguida, são definidos em detalhes e servem como uma especificação do sistema.

2. **Projeto de sistemas e de software** – O processo de projeto de sistemas agrupa os requisitos em sistemas de hardware ou de software. Ele estabelece uma arquitetura do sistema geral. O projeto de software envolve a identificação e a descrição das abstrações fundamentais do sistema de software e suas relações.

3. **Implementação e teste de unidades** – Durante esse estágio, o projeto de software é compreendido como um conjunto de programas ou unidades de programa. O teste de unidades envolve verificar que cada unidade atenda a sua especificação.

4. **Integração e teste de sistemas** – As unidades de programa ou programas individuais são integrados e testados como um sistema completo a fim de garantir que os requisitos de software foram atendidos. Depois dos testes, o sistema de software é entregue ao cliente.

5. **Operação e manutenção** – Normalmente (embora não necessariamente), esta é a fase mais longa do ciclo de vida. O sistema é instalado e colocado em operação. A manutenção envolve corrigir erros que não foram descobertos em estágios anteriores do ciclo de vida, melhorando a implementação das unidades de sistema e aumentando as funções desse sistema à medida que novos requisitos são descobertos.

Um estágio somente pode ser iniciado depois que o estágio anterior tenha sido concluído. Porém, Sommerville (2011, p. 21) afirma que na prática esses estágios acabam se sobrepondo, alimentando uns aos outros de informações. Por exemplo, durante o projeto, os problemas com os requisitos são identificados. O que acontece é que um processo de software não é

Anotações

um modelo linear simples, sequencial, mas envolve iterações entre as fases. Os artefatos de software que são produzidos em cada uma dessas fases podem ser modificados para refletirem todas as alterações realizadas em cada um deles.

Pressman (2011) aponta alguns problemas que podem ser encontrados quando o modelo cascata é aplicado:

1. Os projetos que acontecem nas empresas dificilmente seguem o fluxo sequencial proposto pelo modelo. Alguma iteração sempre ocorre e traz problemas na aplicação do paradigma.

2. Na maioria das vezes, o cliente não consegue definir claramente todas as suas necessidades e o modelo cascata requer essa definição no início das atividades. Portanto, esse modelo tem dificuldade em acomodar a incerteza natural que existe no começo de muitos projetos.

3. Uma versão operacional do sistema somente estará disponível no final do projeto, ou seja, o cliente não terá nenhum contato com o sistema durante o seu desenvolvimento. Isso leva a crer que se algum erro grave não for detectado durante o desenvolvimento, o sistema não atenderá de forma plena as necessidades do cliente.

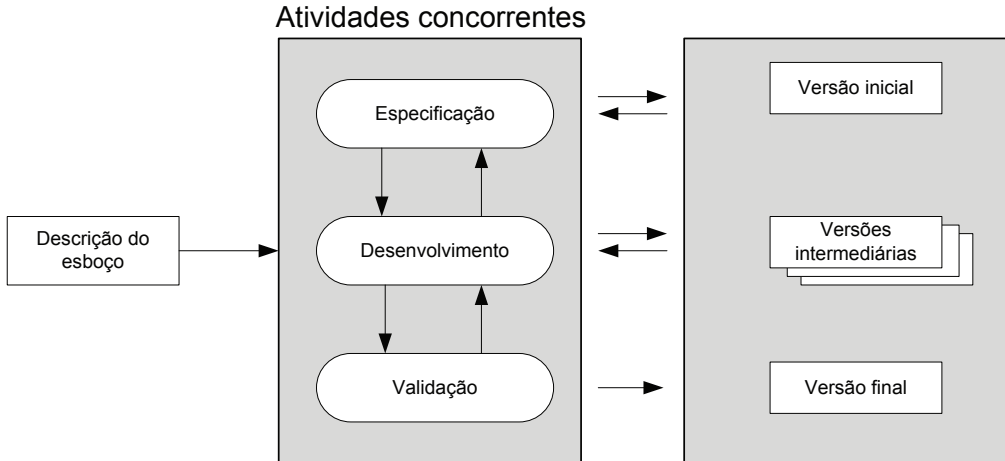
Segundo Sommerville (2011, p. 21) e Pressman (2011, p. 61), o modelo em cascata deve ser utilizado somente quando os requisitos são fixos e tenham pouca probabilidade de serem alterados durante o desenvolvimento do sistema e o trabalho deve ser realizado até sua finalização de forma linear. Sommerville (2011, p.21) ainda afirma que “o modelo cascata reflete o tipo de processo usado em outros projetos de engenharia. Como é mais fácil usar um modelo de gerenciamento comum para todo o projeto, processos de software baseados no modelo em cascata ainda são comumente utilizados”.

<http://www.youtube.com/watch?v=vaavIT2Bqz8>.

Vídeo de demonstração do modelo de desenvolvimento cascata simulado pelo jogo SE•RPG.

Desenvolvimento incremental

O desenvolvimento incremental, segundo Sommerville (2011, p. 21) tem como base a ideia

de desenvolver uma implementação inicial, baseada em uma reunião com os envolvidos para definir os objetivos gerais do software, mostrar para o usuário e fazer seu refinamento por meio de várias versões, até que um sistema adequado tenha sido desenvolvido.

Fonte: Sommerville (2011, p.22)

Assim, as atividades de especificação, desenvolvimento e validação são realizadas concorrentemente com um rápido feedback entre todas as atividades. A cada nova versão, o sistema incorpora novos requisitos definidos pelo cliente.

Para Pressman (2011, p. 63), inicialmente, é necessário desenvolver um projeto rápido que deve se concentrar em uma representação daqueles aspectos do software que serão visíveis aos usuários finais, como, por exemplo, o layout da interface com o usuário.

O desenvolvimento incremental apresenta algumas vantagens importantes em relação ao modelo em cascata. Sommerville (2011, p. 22) coloca três vantagens: (1) se o cliente mudar seus requisitos, o custo será reduzido, pois a quantidade de análise e documentação a ser refeita é menor do que no modelo em cascata; (2) é mais fácil obter um retorno dos clientes sobre o desenvolvimento que foi feito, pois os clientes vão acompanhando o desenvolvimento

do software à medida que novas versões são apresentadas a eles; (3) os clientes podem começar a utilizar o software logo que as versões iniciais forem disponibilizadas, o que não acontece com o modelo cascata.

Entretanto, a partir de uma perspectiva de engenharia e de gerenciamento, existem alguns problemas:

> **1. O processo não** é **visível** ▯ os gerentes necessitam que o desenvolvimento seja regular, para que possam medir o progresso. Se os sistemas são desenvolvidos rapidamente, não é viável produzir documentos que reflitam cada versão do sistema.
>
> **2. Os sistemas frequentemente são mal estruturados** ▯ a mudança constante tende a corromper a estrutura do software. Incorporar modificações no software torna-se cada vez mais difícil e oneroso.
>
> **3. Podem ser exigidas ferramentas e técnicas especiais** ▯ elas possibilitam rápido desenvolvimento, mas podem ser incompatíveis com outras ferramentas ou técnicas, e relativamente poucas pessoas podem ter a habilitação necessária para utilizá-las.

Para sistemas pequenos (com menos de 100 mil linhas de código) ou para sistemas de porte médio (com até 500 mil linhas de código), com tempo de vida razoavelmente curto, a abordagem incremental de desenvolvimento talvez seja a melhor opção. Contudo, para sistemas de grande porte, de longo tempo de vida, nos quais várias equipes desenvolvem diferentes partes do sistema, os problemas de se utilizar o desenvolvimento incremental se tornam particularmente graves. Para esses sistemas, é recomendado um processo misto, que incorpore as melhores características do modelo de desenvolvimento em cascata e do incremental, ou ainda algum outro modelo disponível na literatura.

Na literatura referente a modelos de processo de software pode-se encontrar a prototipação como um exemplo de abordagem incremental.

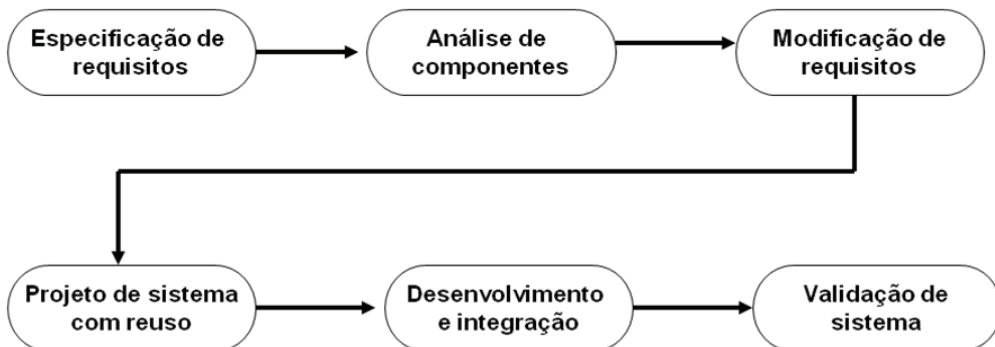
Engenharia de software orientada a reuso

Na maioria dos projetos de software, ocorre algum reuso de software, pois, normalmente,

Anotações

a equipe que trabalha no projeto conhece projetos ou códigos análogos ao que está sendo desenvolvido. Ela busca esses códigos, faz as modificações conforme a necessidade do cliente e os incorporam em seus sistemas. Independentemente do processo de software que está sendo utilizado, pode ocorrer esse reuso informal.

Porém, nos últimos anos, uma abordagem para desenvolvimento de software, com foco no reuso de software existente tem emergido e se tornado cada vez mais utilizada. A abordagem orientada a reuso conta com um grande número de componentes de software reutilizáveis, que podem ser acessados, e com um framework de integração para esses componentes. Às vezes, esses componentes são sistemas propriamente ditos (sistemas COTS – *commercial off-the-shelf* - sistemas comerciais de prateleira), que podem ser utilizados para proporcionar funcionalidade específica, como formatação de textos, cálculo numérico, entre outros (SOMMERVILLE, 2011, p. 23).

O modelo genérico de processo baseado em reuso é mostrado na figura abaixo (SOMMERVILLE, 2007, p.46). Note que, embora as etapas de especificação de requisitos e de validação sejam comparáveis com outros processos, as etapas intermediárias em um processo orientado a reuso são diferentes.

Anotações

Conforme Sommerville (2011, p.23), essas etapas são:

1. **Análise de componentes** – considerando a especificação de requisitos, é feita uma busca de componentes para implementar essa especificação. Pode ser que não sejam encontrados componentes que atendam a toda a especificação de requisitos, ou seja, pode fornecer somente parte da funcionalidade requerida.

2. **Modificação de requisitos** – no decorrer dessa etapa, os requisitos são analisados, levando-se em consideração as informações sobre os componentes que foram encontrados na etapa anterior. Se for possível, os requisitos são então modificados para refletir os componentes disponíveis. Quando isso não for possível, ou seja, quando as modificações forem impossíveis, a etapa de análise de componentes deverá ser refeita, a fim de procurar outras soluções.

3. **Projeto do sistema com reuso** – durante essa etapa, o *framework* do sistema é projetado ou então alguma infraestrutura existente é reutilizada. Os projetistas levam em consideração os componentes que são reusados e organizam o *framework* para tratar desse aspecto. Se os componentes reusáveis não estiverem disponíveis, pode ser necessário que um novo software deva ser projetado.

4. **Desenvolvimento e integração** – nessa etapa, o software que não puder ser comprado deverá ser desenvolvido, e os componentes e sistemas COTS serão integrados, a fim de criar um novo sistema. A integração de sistemas, nessa abordagem, pode ser parte do processo de desenvolvimento, em vez de uma atividade separada.

Deve-se tomar muito cuidado ao utilizar essa abordagem, pois não se tem como evitar as alterações nos requisitos dos usuários e isso pode acabar levando ao desenvolvimento de um sistema que não atenda as suas reais necessidades. Há também o fato de que o controle da evolução do sistema fique comprometido, pois as novas versões dos componentes reusáveis não estão sob o controle da organização que as está utilizando.

De qualquer forma, a abordagem baseada em reuso tem a vantagem de propiciar a entrega mais rápida do software, pois reduz sensivelmente a quantidade de software que a empresa deve desenvolver, reduzindo, consequentemente, os custos de desenvolvimento, bem como os seus riscos.

Anotações

PGDS - Processo de gerenciamento e desenvolvimento de sistemas – DATASUS

Em busca de direcionamento e padronização dos seus processos e da melhoria contínua da qualidade dos produtos e serviços de tecnologia da informação, o DATASUS elaborou suas metodologias de desenvolvimento de software - PDS e de gerenciamento de projetos - EGP. Essas metodologias evoluíram, acompanhando o desenvolvimento tecnológico e as práticas de sucesso dos projetos realizados. Em 2010, com a implantação da Unidade de Apoio a Programas e Projetos (UAPP), o PDS e a EGP foram unificados em uma metodologia, agora denominada Processo de Gerenciamento e Desenvolvimento de Sistemas - PGDS.

Ela foi criada para auxiliar o DATASUS na elaboração, planejamento, execução, controle, monitoramento e encerramento de seus projetos, por meio das melhores práticas de gerenciamento disponíveis no mercado e as já adotadas pelo DATASUS.

Fases do PGDS

O PGDS foi criado para auxiliar o DATASUS/SE no planejamento, execução, controle, monitoramento e encerramento de seus projetos. Serve como um guia para Gestores, Coordenadores, Líderes e equipes de projetos, equipe da UAPP e qualquer outro envolvido nos projetos.

O PGDS é estruturado com base em 4 elementos básicos, que representam “quem” faz “o quê”, “como” e “quando”:

Papéis (quem) - Um papel define o comportamento e responsabilidades de um profissional ou grupo de profissionais que participam do desenvolvimento do projeto. O comportamento é representado através das atividades que cada papel deve desempenhar ao longo do projeto. As responsabilidades normalmente estão associadas aos artefatos que cada papel deve produzir e manter ao longo das atividades que realiza. Na prática, um mesmo papel pode ser desempenhado por mais de uma pessoa, assim como uma mesma pessoa pode assumir vários papéis ao longo do projeto.

Anotações

Artefatos (o quê) - Em sentido amplo, o termo artefato representa um produto concreto produzido, modificado ou utilizado pelas atividades de um processo. O PGDS não inclui todos os artefatos de um projeto de desenvolvimento, mas todos os artefatos obrigatórios descritos no PGDS devem ser elaborados ao longo do projeto. O PGDS disponibiliza modelos (templates) para a maioria de seus artefatos, com o objetivo de orientar e facilitar a sua elaboração.

Atividades (como) - Uma atividade no PGDS representa um conjunto de passos e tarefas que um profissional, que desempenha o papel responsável por aquela atividade, deve executar para gerar algum resultado. As atividades envolvem a produção e modificação de artefatos do projeto.

Fases (quando) - As fases do PGDS apresentam a seqüência e a dependência entre as atividades do projeto ao longo do tempo. As atividades no fluxo são divididas em fases do ciclo de vida do projeto e nos papéis responsáveis pela execução de cada uma.

Disponível em: <http://189.28.128.113/pgds/>. Acesso em: 05 jun. 2012.

Caro(a) aluno(a), o DATASUS é o Departamento de Informática do Sistema Único de Saúde do Ministério da Saúde e você pôde perceber, pelo texto acima, que ele criou um processo de software próprio. É muito interessante conhecer todo o material que eles disponibilizam. Se você quiser conhecer, acesse o endereço <http://189.28.128.113/pgds/>. Ou então acesse o endereço <http://www2.datasus.gov.br/DATASUS/index.php?area=01> para ler mais sobre o DATASUS como um todo. É uma leitura bastante esclarecedora.

Disponível em: <http://189.28.128.113/pgds/>. Acesso em 05 jun. 2012.

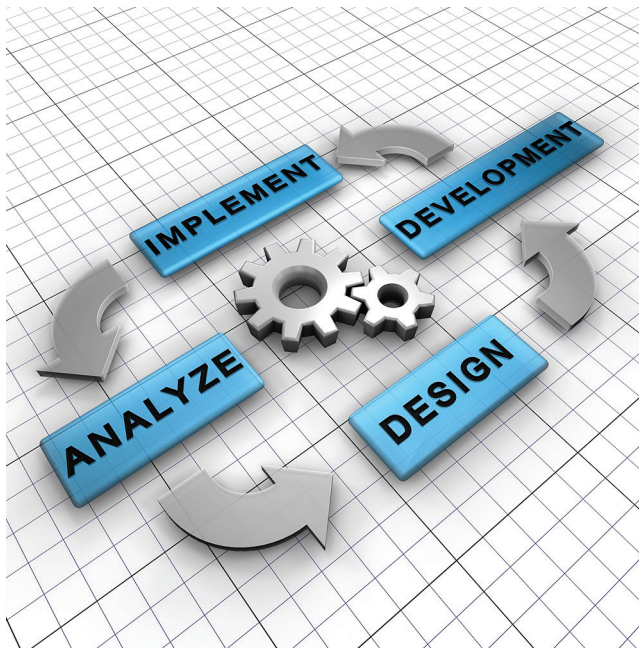
Modelos de Processos de Engenharia de Software

<http://xps-project.googlecode.com/svn-history/r43/trunk/outros/02_Artigo.pdf>.

Anotações

ATIVIDADES BÁSICAS DO PROCESSO DE SOFTWARE

Caro(a) aluno(a), estudando os modelos de processo de software apresentados anteriormente, é possível notar que algumas atividades estão presentes em todos eles, somente, às vezes, essas atividades estão organizadas de forma diferente, dependendo do processo de software que está sendo considerado. Sommerville (2011, p. 24) afirma que no modelo cascata essas atividades são organizadas de forma sequencial, ao passo que no desenvolvimento incremental as mesmas são intercaladas. A forma como essas atividades serão realizadas depende do tipo de software, das pessoas e da organização envolvida.

São quatro as atividades básicas do processo de software: especificação, projeto e implementação, validação e evolução. E a partir de agora, iremos detalhar, de forma genérica, sem considerar um processo de software em específico, cada uma dessas atividades.

Anotações

ESTUDO DO CICLO DE VIDA DO SOFTWARE EM UMA EMPRESA DE DESENVOLVIMENTO DE SISTEMAS

Por Fabrício Luis Marinheiro Lourenço e Márcio Alan Benine

Um estudo do ciclo de vida do software desenvolvido em uma empresa de desenvolvimento de sistemas é importante, pois construir um software com qualidade demanda a utilização e implantação de processos e técnicas de engenharia de software. Este processo é indispensável para que o produto seja entregue ao cliente dentro do prazo e orçamento planejado, alcançando a qualidade esperada. Considerando-se esse procedimento, realizou-se um estudo utilizando-se uma pesquisa bibliográfica, visando encontrar referencial teórico para dar sustentação à questão proposta e um estudo de caso na empresa Nippon Informática Ltda ME. Esta empresa localizada na cidade de Batatais/SP, atua com desenvolvimento de software para as áreas Contábil, Fiscal e Recursos Humanos. No estudo de caso buscou-se elaborar o mapeamento dos processos existentes e após análise dos mesmos, propor um cenário de solução adequado à realidade da empresa. Como resultado final deste estudo, foi apresentada uma proposta de implantação do modelo de ciclo de vida do software, proporcionando a construção do software com qualidade; fornecendo um modelo que atenda aos padrões da engenharia de software e que tenha aspectos de qualidade importantes. Concluiu-se que, cada vez mais, empresas de desenvolvimento de sistemas necessitam adotar processos de software adequados. A definição do ciclo de vida de um software é importante para se ter a visão completa do desenvolvimento do software. Com isto, foi possível definir etapas que abrangem desde a análise dos requisitos até a entrega do software para o cliente.

Fonte: <http://revistas.claretiano.edu.br/index.php/linguagemacademica/article/view/40>. Acesso em: 07 jun. 2012.

ESPECIFICAÇÃO DE SOFTWARE

A especificação de software, ou engenharia de requisitos, é a primeira atividade básica de um processo de software e tem como objetivo definir quais funções são requeridas pelo sistema e

Anotações

identificar as restrições sobre a operação e o desenvolvimento desse sistema. Essa atividade é muito importante e crítica, pois se a definição dos requisitos não for bem realizada, com certeza problemas posteriores no projeto e na implementação do sistema irão acontecer.

Segundo Sommerville (2011, p.24), "o processo de engenharia de requisitos tem como objetivo produzir um documento de requisitos acordados que especifica um sistema que satisfaz os requisitos dos *stakeholders*".

O processo de engenharia de requisitos é composto por quatro fases, conforme descreve Sommerville (2007, p. 50). A unidade seguinte tratará com mais detalhes sobre esse assunto.

1. **Estudo de viabilidade** – uma avaliação é realizada para verificar se as necessidades dos usuários, que foram identificadas, podem ser atendidas utilizando-se as atuais tecnologias de software e hardware, disponíveis na empresa. Esse estudo deve indicar se o sistema proposto será viável, do ponto de vista comercial, e também, se poderá ser desenvolvido considerando restrições orçamentárias, caso as mesmas existam. Um estudo de viabilidade não deve ser caro e demorado, pois é a partir do seu resultado que a decisão de prosseguir com uma análise mais detalhada deve ser tomada.

2. **Levantamento e análise de requisitos** – nesta fase é necessário levantar os requisitos do sistema pela observação de sistemas já existentes, pela conversa com usuários e compradores em potencial, pela análise de tarefas e assim por diante. Essa fase pode envolver o desenvolvimento de um ou mais diferentes modelos e protótipos de sistema. Isso pode ajudar a equipe de desenvolvimento a compreender melhor o sistema a ser especificado.

3. **Especificação de requisitos** – É a atividade de traduzir as informações coletadas durante a fase anterior em um documento que defina um conjunto de requisitos. Tanto os requisitos dos usuários quanto os requisitos de sistema podem ser incluídos nesse documento. De acordo com Sommerville (2011, p.24), os requisitos dos usuários são declarações abstratas dos requisitos do sistema tanto para o cliente quanto para os usuários finais do sistema; os requisitos do sistema são descrições mais detalhadas das funcionalidades a serem fornecidas. Na próxima unidade trataremos com mais detalhes sobre requisitos.

4. **Validação de requisitos** – Essa atividade verifica os requisitos quanto a sua pertinência, consistência e integralidade. Durante o processo de validação, os requisitos que

Anotações

apresentarem problemas devem ser corrigidos, para que a documentação de requisitos fique correta.

As atividades de análise, definição e especificação de requisitos são intercaladas, ou seja, elas não são realizadas seguindo uma sequência rigorosa, pois, com certeza, novos requisitos surgem ao longo do processo.

PROJETO E IMPLEMENTAÇÃO DE SOFTWARE

"O estágio de implementação do desenvolvimento de software é o processo de conversão de uma especificação do sistema em um sistema executável" para Sommerville (2011, p. 25). Esta etapa sempre envolve processos de projeto e programação de software, porém, se uma abordagem incremental de desenvolvimento for utilizada, poderá também envolver o aperfeiçoamento da especificação de software, em que os requisitos foram definidos.

Para Pressman (2011, p. 206), o projeto de software cria uma representação ou modelo do software, fornecendo detalhes sobre a arquitetura do software, as estruturas de dados, interfaces e componentes fundamentais para implementar o sistema. O projeto de software é aplicado independentemente do modelo de processo de software que está sendo utilizado, ou seja, se estiver sendo utilizado o modelo em cascata ou a abordagem incremental.

O início do projeto se dá assim que os requisitos tiverem sido analisados e modelados, ou seja, assim que a modelagem do sistema tiver sido realizada. Com base no documento de requisitos, o engenheiro de software, na fase de modelagem do sistema, deverá elaborar os diagramas da UML – *Unified Modeling Language* (como, por exemplo, o Diagrama de Caso de Uso e o Diagrama de Classes). Na fase de projeto do sistema, o engenheiro de software deverá: i) definir o Diagrama Geral do Sistema; ii) elaborar as Interfaces com o Usuário (telas e relatórios) e iii) desenvolver um conjunto de especificações de casos de uso, sendo que, essas especificações devem conter detalhes suficientes para permitir a codificação. Geralmente,

durante o projeto, o analista de sistemas terá que definir cada componente do sistema ao nível de detalhamento que se fizer necessário para a sua implementação em uma determinada linguagem de programação.

A programação, normalmente começa como era de se esperar, quando termina a atividade de projeto. A programação, ou fase de implementação de um projeto típico envolve a escrita de instruções em Java, C++, C# ou em alguma outra linguagem de programação para implementar o que o analista de sistemas modelou na etapa de projeto. Sendo uma atividade pessoal, não existe um processo geral que seja normalmente seguido durante a programação do sistema. Alguns programadores começarão com os componentes que eles compreendem melhor, passando depois para os mais complexos. Outros preferem deixar os componentes mais fáceis para o fim, porque sabem como desenvolvê-los. Alguns desenvolvedores gostam de definir os dados no início do processo e, então, utilizam esses dados para dirigir o desenvolvimento do programa; outros deixam os dados sem especificação enquanto for possível.

De acordo com Sommerville (2011, p. 27), normalmente, os programadores testam os códigos fontes que eles mesmos desenvolveram, a fim de descobrir defeitos que devem ser removidos. Esse processo é chamado de depuração. O teste e a depuração dos defeitos são processos diferentes. O teste estabelece a existência de defeitos, enquanto que a depuração se ocupa em localizar e corrigir esses defeitos.

Em um processo de depuração, os defeitos no código devem ser localizados, e o código precisa ser corrigido, a fim de cumprir os requisitos. A fim de garantir que a mudança foi realizada corretamente, os testes deverão ser repetidos. Portanto, o processo de depuração é parte tanto do desenvolvimento quanto do teste do software.

VALIDAÇÃO DE SOFTWARE

A validação de software dedica-se a mostrar que um sistema atende tanto as especificações

Anotações

relacionadas no documento de requisitos, quanto às expectativas dos seus usuários. A principal técnica de validação, de acordo com Sommerville (2011, p. 27), é o teste de programa, em que o sistema é executado com dados de testes simulados.

Os testes somente podem ser realizados como uma unidade isolada se o sistema for pequeno. Caso contrário, se o sistema for grande e constituído a partir de subsistemas, que, por sua vez, são construídos partindo-se de módulos, o processo de testes deve evoluir em estágios, ou seja, devem ser realizados de forma incremental, iterativa.

Sommerville (2011, p.27) propõe um processo de teste em três estágios. O primeiro estágio é o teste de componente, em seguida, o sistema integrado é testado e, por fim, o sistema é testado com dados reais, ou seja, com dados do próprio cliente. Idealmente, os defeitos de componentes são descobertos no início do processo, e os problemas de interface são encontrados quando o sistema é integrado.

Os estágios do processo de testes, conforme Sommerville (2011, p. 27) são:

1. **Testes de desenvolvimento** – Para garantir que os componentes individuais estão operando corretamente, é necessário testá-los, de forma independente dos outros componentes do sistema.
2. **Testes de sistema** – Os componentes são integrados para constituírem o sistema. Esse processo se dedica a encontrar erros que resultem de interações não previstas entre os componentes e de problemas com a interface do componente. O teste de sistema também é utilizado para validar que o sistema atende os requisitos funcionais e não funcionais definidos no documento de requisitos.
3. **Teste de aceitação** – Nesse estágio, o sistema é testado com dados reais fornecidos pelo cliente, podendo mostrar falhas na definição de requisitos, pois os dados reais podem exercitar o sistema de modo diferente dos dados de teste.

Anotações

EVOLUÇÃO DE SOFTWARE

Depois que um software é colocado em funcionamento, ou seja, depois que o mesmo é implantado, com certeza ocorrerá mudanças que levarão à alteração desse software. Essas mudanças podem ser, de acordo com Pressman (2011, p. 662), para correção de erros não detectados durante a etapa de validação do software, quando há adaptação a um novo ambiente, quando o cliente solicita novas características ou funções ou ainda quando a aplicação passa por um processo de reengenharia para proporcionar benefício em um contexto moderno.

Sommerville (2011, p. 29) coloca que, historicamente, sempre houve uma fronteira entre o processo de desenvolvimento de software e o processo de evolução desse mesmo software (manutenção de software). O desenvolvimento de software é visto como uma atividade criativa, em que o software é desenvolvido a partir de um conceito inicial até chegar ao sistema em operação. Depois que esse sistema entrou em operação, inicia-se a manutenção de software, no qual o mesmo é modificado. Normalmente, os custos de manutenção são maiores do que os custos de desenvolvimento inicial, mas os processos de manutenção são considerados menos desafiadores do que o desenvolvimento de software original, ainda que tenha um custo mais elevado.

Porém, atualmente, os estágios de desenvolvimento e manutenção têm sido considerados como integrados e contínuos, em vez de dois processos separados. Tem sido mais realista pensar na engenharia de software como um processo evolucionário, em que o software é sempre mudado ao longo de seu período de vida, em resposta a requisitos em constante modificação e às necessidades do cliente.

Anotações

CONSIDERAÇÕES FINAIS

Chegamos ao final de mais uma unidade. Nesta segunda unidade você conheceu o que é um processo de software e também alguns modelos de processo de software.

Um processo de software é um conjunto de atividades com resultados (artefatos) associados a cada uma delas que leva à produção de um software. Todo software deve ser especificado, projetado, implementado e validado. E, após o seu uso pelo usuário, passa por evoluções. Todas essas etapas são muito importantes, mas vimos que a especificação do software é uma etapa imprescindível nesse conjunto, pois, se os requisitos não forem esclarecidos, bem especificados, no início do desenvolvimento, há uma grande chance do software não atender às necessidades do cliente. No tempo que trabalhei com desenvolvimento de sistemas vi isso acontecer algumas vezes. E sabem o que acontece? O usuário acaba não utilizando o sistema e assim o sistema acaba não atingindo o seu objetivo. Na próxima unidade vamos tratar o assunto Requisitos de Software mais detalhadamente, justamente pela importância que mencionei acima.

Após os requisitos estarem declarados e validados, vimos que o projeto do sistema deve ser realizado. Nessa etapa, o sistema é modelado de forma bem detalhada, pois a próxima etapa é a implementação do software. Na unidade quatro trataremos com mais detalhes sobre a modelagem do sistema, em especial sobre a linguagem de modelagem unificada (UML – *Unified Modeling Language*). A implementação é a escrita do sistema em uma linguagem de programação. Nesta disciplina veremos somente a parte teórica relacionada à implementação, pois a parte prática faz parte de outras disciplinas do seu curso.

Mas, afinal, qual a diferença entre processo de software e modelo de processo de software? Um processo de software é o conjunto de atividades (mencionadas acima) e um modelo de processo de software é uma representação abstrata de um processo de software, ou seja, define a sequência em que as atividades do processo serão realizadas.

Existem vários modelos de processo de software descritos na literatura, porém nesta unidade vimos somente alguns desses modelos. O primeiro foi o Modelo em Cascata, que representa as atividades do processo (especificação, desenvolvimento, validação e evolução) como fases separadas, onde uma só pode acontecer depois que a anterior tenha sido concluída. O segundo modelo foi o Desenvolvimento Incremental, que tem como base a ideia de desenvolver uma

implementação inicial, expor ao comentário do usuário/cliente e fazer seu aprimoramento por meio de muitas versões, até que um sistema adequado tenha sido desenvolvido. Nesse modelo, em vez de ter as atividades de especificação, desenvolvimento e validação em separado, todo esse trabalho é realizado concorrentemente. O último modelo que estudamos foi a Engenharia de Software Baseada em Reuso, que baseia-se na existência de um número significativo de componentes reusáveis, sendo que o processo de desenvolvimento de sistemas se concentra na integração desses componentes em um sistema, em vez de partir do zero.

Mas, afinal, qual o melhor modelo de processo de software para uma empresa? Infelizmente a resposta para essa pergunta não é tão simples. Não existe um processo ideal de software e os modelos não são mutuamente exclusivos e na maioria das vezes, podem ser usados em conjuntos, em especial para o desenvolvimento de sistemas de grande porte.

O aumento na demanda por software de qualidade vem causando grande pressão sobre as empresas que trabalham com desenvolvimento de software. As entregas de software obedecendo ao cronograma e custos previstos vêm se tornando, a cada dia, um diferencial importante nesse ramo de atividade. Por isso, as empresas procuram por processos de software que propiciem o desenvolvimento de produtos com qualidade, e que respeitem o custo e cronograma previstos.

Na próxima unidade vamos conhecer um pouco mais sobre requisitos de software e entender por que os requisitos são tão importantes em um processo de software.

ATIVIDADE DE AUTOESTUDO

1. Faça um comparativo entre os Modelos de Processo de Software – Modelo Cascata e Desenvolvimento Incremental.
2. Explique cada uma das atividades básicas que compõem um processo de software. Essas atividades devem ser realizadas sempre na ordem descrita nesta unidade? Justifique sua resposta.
3. Considerando os modelos de processo de software apresentados nesta unidade, defina um modelo de processo de software que poderia ser utilizado por uma pequena empresa de desenvolvimento de sistemas.

UNIDADE III

REQUISITOS DE SOFTWARE

Professora Me. Márcia Cristina Dadalto Pascutti

Objetivos de Aprendizagem

- Entender os diversos tipos de requisitos relacionados ao desenvolvimento de software.
- Expor a importância do documento de requisitos.
- Compreender o processo de engenharia de requisitos.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- **Tipos de Requisitos de Software**
- **Documento de Requisitos**
- **Engenharia de Requisitos**

INTRODUÇÃO

Caro(a) aluno(a), na segunda unidade você aprendeu os conceitos relacionados a processo de software e viu que um processo é composto de quatro atividades fundamentais: especificação de software, projeto e implementação de software, validação de software e, finalmente, evolução de software.

Esta unidade vai tratar especificamente sobre requisitos de software e, no final desta unidade você vai compreender por que os requisitos são importantes e devem ser muito bem definidos para que o software desenvolvido alcance seus objetivos.

Uma das tarefas mais difíceis que os desenvolvedores de software enfrentam é entender os requisitos de um problema. Os requisitos definirão o que o sistema deve fazer, suas propriedades emergentes desejáveis e essenciais e as restrições quanto à operação do sistema. Essa definição de requisitos somente é possível com a comunicação entre os clientes e os usuários de software e os desenvolvedores de software.

As preferências, preconceitos e recusas dos usuários, além das questões políticas e organizacionais influenciam diretamente nos requisitos do sistema, portanto, a engenharia de software não é simplesmente um processo técnico (SOMMERVILLE, 2007, p. 78).

Anotações

Nesta unidade, você aprenderá a diferença entre os vários tipos de requisitos e trataremos, principalmente, dos requisitos funcionais e não funcionais. Os requisitos funcionais representam as descrições das diversas funções que clientes e usuários querem ou precisam que o software ofereça. Um exemplo de requisito funcional é “o sistema deve possibilitar o cadastramento dos dados pessoais dos pacientes”. Já, os requisitos não funcionais, declaram as restrições ou atributos de qualidade para um software, como, por exemplo, precisão, manutenibilidade, usabilidade entre outros. “O tempo de desenvolvimento não deve ultrapassar seis meses” é um exemplo de requisito não funcional.

Todos os requisitos definidos, sejam eles funcionais ou não funcionais, devem estar escritos em um documento de requisitos, que servirá como base para todas as atividades subsequentes do desenvolvimento e também fornecerá um ponto de referência para qualquer validação do software construído.

Também estudaremos nesta unidade sobre os requisitos de qualidade, que são definidos pela Norma ISO/IEC 9126 e que também devem ser considerados quando um software está sendo projetado.

E, por fim, veremos que a engenharia de requisitos é um processo que envolve quatro atividades genéricas: avaliar se o sistema que está sendo projetado será útil para a empresa (estudo de viabilidade), obter e analisar os requisitos (levantamento e análise), especificar esses requisitos, convertendo-os em um documento de requisitos (especificação de requisitos) e, finalmente, verificar se os requisitos realmente definem o sistema que o cliente deseja (validação).

Anotações

REQUISITOS DE SOFTWARE

Normalmente, os problemas que os desenvolvedores de software têm para solucionar são, muitas vezes, imensamente complexos e se o sistema for novo, entender a natureza desses problemas pode ser muito mais difícil ainda. As descrições das funções e das restrições são os requisitos para o sistema; e o processo de descobrir, analisar, documentar e verificar essas funções e restrições é chamado de engenharia de requisitos.

De acordo com Sommerville (2011, p. 57), a indústria de software não utiliza o termo requisito de modo consistente. Muitas vezes, o requisito é visto como uma declaração abstrata em alto nível, de uma função que o sistema deve fornecer ou de uma restrição do sistema. Em outras vezes, ele é uma definição detalhada e formal, de uma função do sistema.

Alguns dos problemas que surgem durante a especificação de requisitos são as falhas em não fazer uma separação clara entre os diferentes níveis de descrição dos requisitos. Por isso, Sommerville (2011, p. 57) propõe uma distinção entre eles por meio do uso do termo “requisitos de usuário”, para expressar os requisitos abstratos de alto nível, e “requisitos de sistema”, para expressar a descrição detalhada que o sistema deve fazer. Dessa forma, os requisitos de

Anotações

usuário deverão fornecer, em forma de declarações, quais serviços o sistema deverá oferecer e as restrições com as quais o sistema deve operar. Já os requisitos de sistema são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema.

Caro(a) aluno(a), se sua empresa deseja estabelecer um contrato para o desenvolvimento de um grande sistema, ela deve definir todas as necessidades/requisitos de maneira suficientemente abstrata para que uma solução não seja predefinida, ou seja, essas necessidades devem ser redigidas de modo que os diversos fornecedores possam apresentar propostas, oferecendo, talvez, diferentes maneiras de atender às necessidades organizacionais da sua empresa. Uma vez estabelecido um contrato, o fornecedor precisa preparar uma definição de sistema para o cliente, com mais detalhes, de modo que o cliente compreenda e possa validar o que o software fará. Esses dois documentos podem ser chamados de documentos de requisitos do sistema. Veremos mais adiante o documento de requisitos com mais detalhes.

Mas, e o que pode acontecer se os requisitos não forem definidos corretamente, se ficarem errados? Se isso acontecer, o sistema pode não ser entregue no prazo combinado e com o custo acima do esperado no início do projeto; o usuário final e o cliente não ficarão satisfeitos com o sistema e isso pode até implicar no descarte do sistema. Portanto, o ideal é que essa etapa seja muito bem elaborada.

Reflita!

"A parte mais difícil ao construir um sistema de software é decidir o que construir. Nenhuma parte do trabalho afeta tanto o sistema resultante se for feita a coisa errada. Nenhuma outra parte é mais difícil de consertar depois".

Fred Brooks, Engenheiro de Software.

REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Primeiramente, vamos definir o que é requisito, independentemente da área de informática. Um requisito é a condição imprescindível para a aquisição ou preenchimento de determinado objetivo. Na abordagem da engenharia de software, segundo Sommerville (2011, p. 57), “os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento”. Esses requisitos dizem respeito às necessidades dos usuários para um sistema que deve atender um determinado objetivo, como, por exemplo, cadastrar um pedido de venda ou emitir um relatório. A engenharia de requisitos é um processo que engloba as atividades que são necessárias para criar e manter um documento de requisitos de sistema. Essas atividades são: estudo de viabilidade, levantamento e análise de requisitos, especificação de requisitos e, finalmente, a validação desses requisitos.

De acordo com Sommerville (2011, p. 59), os requisitos de software são, normalmente, classificados como funcionais ou não funcionais:

1. **Requisitos funcionais** define as funções que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer. Exemplos de requisitos funcionais: o software deve possibilitar o cálculo das comissões dos vendedores de acordo com os produtos vendidos; o software deve emitir relatórios de compras e vendas por período; o sistema deve mostrar, para cada aluno, as disciplinas em que o mesmo foi aprovado ou reprovado.
2. **Requisitos não funcionais** são os requisitos relacionados à utilização do software em termos de desempenho, confiabilidade, segurança, usabilidade e portabilidade entre outros. Exemplos de requisitos não funcionais: o sistema deve ser protegido para acesso apenas de usuários autorizados; o tempo de resposta do sistema não deve ultrapassar 20 segundos; o tempo de desenvolvimento não deve ultrapassar doze meses.

Contudo, a diferenciação entre esses dois tipos de requisitos não é tão clara como sugerem as definições acima. Um requisito referente à proteção, pode parecer ser um requisito não

Anotações

funcional. Porém, quando desenvolvido com mais detalhes, pode levar a outros requisitos que são claramente funcionais, como a necessidade de incluir recursos de autorização de usuários no sistema (SOMMERVILLE, 2011, p. 59). Portanto, embora seja interessante separar os requisitos em funcionais e não funcionais, devemos lembrar que essa é, na verdade, uma distinção artificial. O que é muito importante é que os requisitos, sejam eles funcionais ou não funcionais, sejam claramente definidos.

Requisitos funcionais

Os requisitos funcionais devem descrever detalhadamente os serviços e a funcionalidade que devem ser fornecidas pelo sistema, indicando suas entradas e saídas, exceções etc. Esses requisitos podem ser expressos de diversas maneiras, com diferentes níveis de detalhes. A imprecisão na especificação de requisitos é uma das causas de muitos problemas da engenharia de software (SOMMERVILLE, 2011, p. 60). Pode acontecer que um desenvolvedor de sistemas interprete um requisito ambíguo para simplificar sua implementação, porém, nem sempre é isso o que o cliente quer. E quando isso acontece, pode ser que novos requisitos devam ser estabelecidos, sendo necessário realizar mudanças no sistema, podendo atrasar a entrega final do sistema e, consequente, aumento de custos.

De acordo com Sommerville (2011, p. 60), em princípio, a especificação de requisitos funcionais de um sistema deve ser completa e consistente. A completeza denota que todas as funções requeridas pelo usuário devem estar definidas e a consistência denota que os requisitos não devem ter definições contraditórias. Na prática, para grandes sistemas, atingir a consistência e a completeza dos requisitos é bastante difícil, por causa da complexidade inerente ao sistema e, em parte, porque diferentes pontos de vista apresentam necessidades inconsistentes.

Requisitos não funcionais

Os requisitos não funcionais são aqueles que não dizem respeito diretamente às funções específicas oferecidas pelo sistema. Eles podem estar relacionados a propriedades, como

Anotações

confiabilidade, tempo de resposta e espaço em disco. Como alternativa, eles podem definir restrições para o sistema, como a capacidade dos dispositivos de E/S (entrada/saída) e as representações de dados utilizadas nas interfaces de sistema (SOMMERVILLE, 2011, p. 60).

Os requisitos não funcionais surgem conforme a necessidade dos usuários, em razão de restrições de orçamento, de políticas organizacionais, pela necessidade de interoperabilidade com outros sistemas de software ou hardware ou devido a fatores externos, como por exemplo, regulamentos de segurança e legislação sobre privacidade.

Sommerville (2011, p.61) faz uma classificação dos requisitos não funcionais em requisitos de produto, requisitos organizacionais e requisitos externos. Os requisitos de produto são aqueles que especificam o comportamento do produto, podendo ser subdivididos em requisitos de usabilidade, de eficiência, de confiança e de proteção. Os requisitos organizacionais são aqueles derivados das políticas e procedimentos da organização do cliente e do desenvolvedor e são subdivididos em requisitos ambientais, operacionais e de desenvolvimento. Finalmente, os requisitos externos abrangem todos os requisitos que procedem de fatores externos ao sistema e seu processo de desenvolvimento e são subdivididos em requisitos reguladores, éticos e legais.

Os requisitos funcionais e não funcionais deveriam ser diferenciados em um documento de requisitos, porém, na prática, não é fácil fazer essa distinção. Em nossos documentos de requisitos nos preocuparemos mais com os requisitos funcionais do sistema. Se os requisitos não funcionais forem definidos separadamente dos requisitos funcionais, pode ser difícil enxergar a relação existente entre eles. Se eles forem definidos com os requisitos funcionais, poderá ser difícil separar considerações funcionais e não funcionais e identificar os requisitos que correspondem ao sistema como um todo. É preciso encontrar um equilíbrio adequado e isso depende do tipo de sistema que está sendo modelado. Contudo, requisitos claramente relacionados às propriedades emergentes do sistema devem ser explicitamente destacados. Isso pode ser feito colocando-os em uma seção separada do documento de requisitos ou diferenciando-os, de alguma maneira, dos outros requisitos de sistema.

Anotações

____

____

____

____

REQUISITOS DE USUÁRIO

De acordo com Sommerville (2007, p. 85), os requisitos de usuários para um sistema devem descrever os requisitos funcionais e não funcionais de forma que usuários do sistema que não tenham conhecimentos técnicos detalhados consigam entender. Eles devem especificar somente o comportamento externo do sistema, evitando sempre que possível as características do projeto de sistema. Portanto, não devem ser definidos utilizando-se um modelo de implementação, e sim, escritos com o uso de linguagem natural, formulários e diagramas intuitivos simples.

REQUISITOS DE SISTEMA

Os requisitos de sistema são descrições mais detalhadas dos requisitos do usuário, servindo como base para um contrato destinado à implementação do sistema e, portanto, devem ser uma especificação completa e consistente de todo o sistema (SOMMERVILLE, 2007, p. 87). Eles são utilizados pelos engenheiros de software como ponto de partida para o projeto de sistema.

Antes de qualquer coisa, os requisitos de sistema deveriam definir o que o sistema deveria fazer, e não como ele teria de ser implementado, porém, no que se refere aos detalhes exigidos para especificar o sistema completamente, é quase impossível excluir todas as informações de projeto. Há, pelo menos, duas razões para isso:

1. Uma arquitetura inicial do sistema pode ser definida para ajudar a estruturar a especificação de requisitos.
2. Na maioria dos casos, os sistemas devem interoperar com outros sistemas existentes, restringindo assim o projeto em desenvolvimento, sendo que, muitas vezes, essas restrições geram requisitos para o novo sistema.

De acordo com Sommerville (2011, p. 58), os requisitos devem ser escritos em níveis

Anotações

diferentes de detalhamento para que diferentes leitores possam usá-los de formas diferentes. Os possíveis leitores para os requisitos de usuário são: gerentes clientes, usuários finais do sistema, engenheiros clientes, gerentes contratantes e arquitetos de software. Esses leitores não tem a preocupação com a forma como o sistema será implementado. Já para os requisitos de sistema podem-se ter os seguintes leitores: usuários finais do sistema, engenheiros clientes, arquitetos de sistema e desenvolvedores de software. Esses leitores precisam saber com mais detalhes o que o sistema fará, principalmente os desenvolvedores que estarão envolvidos no projeto e na implementação do sistema.

"As sementes das principais catástrofes de software são normalmente semeadas nos três primeiros meses do projeto de software"

Caper Jones, Especialista em Engenharia de Software.

O DOCUMENTO DE REQUISITOS DE SOFTWARE

O documento de requisitos de software ou especificação de requisitos de software é a declaração oficial do que é exigido dos desenvolvedores de sistema. Ele deve incluir os requisitos de usuários para um sistema e uma especificação detalhada dos requisitos de sistema. Em alguns casos, os requisitos de usuário e de sistema podem ser integrados em uma única descrição. Em outros casos, os requisitos de usuário são definidos em uma introdução à especificação dos requisitos de sistema. Se houver um grande número de requisitos, os requisitos detalhados de sistema poderão ser apresentados como documentos separados.

O documento de requisitos serve como um termo de consenso entre a equipe técnica

Anotações

(desenvolvedores) e o cliente e constitui a base para as atividades subsequentes do desenvolvimento do sistema, fornecendo um ponto de referência para qualquer validação futura do software construído. Além disso, o documento de requisitos estabelece o escopo (o que faz parte e o que não faz parte) do sistema, abrangendo um conjunto diversificado de usuários, que vai desde a alta gerência da organização, que está pagando pelo sistema, até os engenheiros responsáveis pelo desenvolvimento do software.

A tabela abaixo mostra uma possível organização de um documento de requisitos definido por Sommerville (2011, p. 64), baseada em uma norma IEEE (*Institute of Electrical and Electronics Engineers*) para documentos de requisitos.

Tabela – A estrutura de um documento de requisitos

Capítulo	Descrição
Prefácio	Deve definir os possíveis leitores do documento e descrever seu histórico de versões, incluindo uma justificativa para a criação de uma nova versão e um resumo das mudanças feitas em cada versão.
Introdução	Deve descrever a necessidade para o sistema. Deve descrever brevemente as funções do sistema e explicar como ele vai funcionar com outros sistemas. Também deve descrever como o sistema atende aos objetivos globais de negócio ou estratégicos da organização que encomendou o software.
Glossário	Deve definir os termos técnicos usados no documento. Você não deve fazer suposições sobre a experiência ou o conhecimento do leitor.
Definição de requisitos de usuário	Deve descrever os serviços fornecidos ao usuário. Os requisitos não funcionais de sistema também devem ser descritos nessa seção. Essa descrição pode usar a linguagem natural, diagramas ou outras notações compreensíveis para os clientes. Normas de produto e processos a serem seguidos devem ser especificados.
Arquitetura do sistema	Deve apresentar uma visão geral em alto nível da arquitetura do sistema previsto, mostrando a distribuição de funções entre os módulos do sistema. Componentes de arquitetura que são reusados devem ser destacados.

Anotações

Capítulo	Descrição
Especificação de requisitos do sistema	Deve descrever em detalhes os requisitos funcionais e não funcionais. Se necessário, também podem ser adicionados mais detalhes aos requisitos não funcionais. Interfaces com outros sistemas podem ser definidas.
Modelos do sistema	Pode incluir modelos gráficos do sistema que mostram os relacionamentos entre os componentes do sistema, o sistema e seu ambiente. Exemplos de possíveis modelos são modelos de objetos, modelos de fluxo de dados ou modelo semânticos de dados.
Evolução do sistema	Deve descrever os pressupostos fundamentais em que o sistema se baseia, bem como quaisquer mudanças previstas, em decorrência da evolução do hardware, de mudanças nas necessidades do usuário, etc. Essa seção é útil para projetistas de sistema, pois pode ajudá-los a evitar decisões capazes de restringir possíveis mudanças futuras no sistema.
Apêndices	Deve fornecer informações detalhadas e específicas relacionadas à aplicação em desenvolvimento, além de descrições de hardware e banco de dados, por exemplo. Os requisitos de hardware definem as configurações mínimas ideais para o sistema. Requisitos de banco de dados definem a organização lógica dos dados usados pelo sistema e os relacionamentos entre esses dados.
Índice	Vários índices podem ser incluídos no documento. Pode haver, além de um índice alfabético normal, um índice de diagramas, de funções, entre outros pertinentes.

Fonte: (SOMMERVILLE, 2011, p.64)

Para o desenvolvimento da nossa disciplina, usarei modelos de documento de requisitos mais simplificados do que o apresentado na tabela acima. O documento de requisitos trará detalhes de como o sistema funciona atualmente e quais funcionalidades o usuário deseja para o novo sistema. Abaixo segue um modelo de documento de requisitos para uma locadora de filmes. Lembre-se que deve ser a partir do documento de requisitos que faremos a modelagem do sistema, que será detalhada na próxima unidade.

Exemplo de Documento de Requisitos – Locadora de Filmes

Uma determinada locadora possui muitos títulos em seu acervo e não consegue controlar

Anotações

de maneira eficiente as locações, devoluções e reservas dos filmes. Portanto, ela deseja ter um sistema informatizado que controle todas as locações, devoluções e reservas de maneira eficiente, para aperfeiçoar o seu atendimento com o cliente e seu controle interno.

Atualmente, a locadora possui uma ficha para o cadastro de clientes com os seguintes dados: nome do cliente, fone residencial, fone celular, sexo, RG, CPF, endereço completo, data de nascimento, estado civil e nomes de cinco dependentes e o grau de parentesco de cada dependente (o dependente pode locar filmes em nome do cliente).

O sistema informatizado deve:

1. **Manter o cadastro de filmes.** Neste cadastro deverá conter os seguintes dados: nome do filme, duração, sinopse, classificação, gênero, diretor, elenco. Para cada cópia do filme é necessário saber o fornecedor da mesma, a data da compra, o valor pago e o tipo (VHS ou DVD).
2. **Controlar locações**

- A locação é feita mediante a verificação de cadastro do cliente. Se o cliente for cadastrado então se efetua a locação, se não se cadastra o cliente.
- Caso a locação seja efetuada pelo dependente do cliente, é necessário deixar registrado qual o dependente e qual o cliente.
- É verificado se o filme está disponível e se o cliente possui pendências financeiras ou atraso de devolução, caso uma das alternativas seja afirmativa bloqueia-se a operação, sendo liberada somente após a devida regularização.
- Emitir comprovante de locação com a data prevista para devolução de cada filme, discriminação dos filmes e se o pagamento foi ou não efetuado.
- A data prevista para devolução deve ser calculada desconsiderando domingos e feriados. Cada categoria pode ter um prazo diferente para que o cliente possa ficar com o filme. Por exemplo: a categoria LANÇAMENTO permite que o cliente fique com o filme por 2 dias.

3. Controlar devoluções

- Verificar se a devolução está no prazo correto e se o pagamento foi efetuado, caso

Anotações

o prazo esteja vencido calcular a multa incidente. Efetuado o pagamento emite-se o recibo de devolução.

- Não esquecer que não pode ser cobrada multa caso seja domingo ou feriado.

4. **Controlar reservas**

- Verificar se o cliente já está cadastrado, caso contrário o sistema permite o cadastro do cliente no momento da reserva. Também é verificado se o filme desejado está disponível para reserva.
- Reservar somente para clientes sem pendências financeiras e devoluções vencidas.

5. **Consultar Filmes Locados por Cliente**: o sistema deve ter uma consulta em que seja informado um determinado cliente e sejam mostrados todos os filmes já locados por esse cliente e mostre também a data em que cada filme foi locado.

6. **Consultar Reservas por Filme**: o sistema deve ter uma consulta em que seja informado um determinado filme e sejam mostradas todas as reservas efetuadas para aquele filme, no período informado.

7. **Emitir os seguintes relatórios**

- **Relatório Geral de Clientes**, onde conste o código, nome, endereço, telefone e dependentes do cliente.
- **Etiquetas com códigos de barras** para a identificação das cópias no processo de locação e devolução.
- **Relatório de filmes por gênero**, onde conste o código do filme, o nome do filme, o nome do diretor do filme, os nomes dos atores do filme, o total de cópias, o total de cópias locadas e o total de cópias disponíveis. O relatório deve ser agrupado por gênero, mostrando também o código e a descrição do gênero.
- **Relatório de filmes locados por cliente por período**. Para cada cliente devem ser emitidas todas as cópias que estão locadas para ele. Deve sair no relatório: o código e o nome do cliente, o código do filme, o nome do filme, o código da cópia (exemplar), a data de locação e o valor da locação. O relatório deve ser agrupado por cliente e devem sair somente as cópias locadas e não devolvidas.
- **Relatório de cópias não devolvidas**, onde conste o código do filme, o nome do filme, o código da fita, o nome do cliente, o telefone do cliente, a data de locação, a data prevista para devolução e o número de dias em atraso.

Anotações

___

___

___

___

- **Relatório dos filmes mais locados**, onde conste o código do filme, o nome do filme, a descrição do gênero e o número total de locações. O relatório deve ser agrupado por mês/ano, ou seja, para um determinado mês/ano, devem ser emitidos os 10 (dez) filmes mais locados.
- **Relatório de Reservas por período**, onde conste o código do cliente, o nome do cliente, o telefone do cliente, o código do filme reservado, o nome do filme, a data em que foi feita a reserva (data em que o cliente telefonou para a locadora dizendo que queria fazer a reserva).
- **Relatório dos valores das locações mensais.** Deverá mostrar os valores das locações de determinado mês, separado por data e somatória de valores de cada dia, somando-se assim ao final, uma totalidade de locações. Nele deve-se conter a data e a soma das locações desta data.

Todos os relatórios servirão para o processo de tomadas de decisões nos quais os Administradores poderão obter informações sobre o andamento da locadora.

REQUISITOS DE QUALIDADE

Quanto mais rígidos os requisitos de qualidade e mais complexo o software a ser desenvolvido, aumenta-se a necessidade de se aplicar teorias e ferramentas que garantam que esses requisitos sejam satisfeitos. A Norma ISO (*The International Organization for Standardization)* / IEC (*The International Electrotechnical Commission*) 9126 define seis características de qualidade de software que devem ser avaliadas:

Funcionalidade é a capacidade de um software fornecer funcionalidades que atendam as necessidades explícitas e implícitas dos usuários, dentro de um determinado contexto de uso.

Usabilidade conjunto de atributos que evidenciam o esforço necessário para a utilização do software.

Confiabilidade indica a capacidade do software em manter seu nível de desempenho sob determinadas condições durante um período de tempo estabelecido.

Anotações

Eficiência – indica que o tempo de execução e os recursos envolvidos são compatíveis com o nível de desempenho do software.

Manutenibilidade – conjunto de atributos que evidenciam o esforço necessário para fazer modificações especificadas no software, incluindo tanto as melhorias/ extensões de funcionalidades quanto as correções de defeitos, falhas ou erros.

Portabilidade – indica a capacidade do software de ser transferido de um ambiente para outro.

A ISO/IEC formam o sistema especializado para padronização mais conhecido no mundo. Você pode obter mais informações através do endereço <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749>. Existe também adequação dessa norma para o Brasil – a NBR ISO/IEC 9126-1. Verifique no endereço <http://www.abnt.org.br/>.

Especificação de requisitos no desenvolvimento de software para TV Digital Interativa no Brasil: Reflexões e Relato de Experiência

Por Carlos Eduardo Marquioni

O processo de implantação da TV Digital no Brasil iniciou uma nova fase em 2009, relacionada à definição de mecanismos para interação através do televisor. Contudo, além de viabilizar a infraestrutura tecnológica para troca de informações entre o telespectador e os difusores, parece relevante considerar a utilização de processos de software para que os produtos desenvolvidos que possibilitam a interação tenham qualidade. Este trabalho apresenta conceitos do processo de Especificação da Engenharia de Requisitos e relaciona boas práticas de escrita para gerar requisitos textuais que, integradas com a técnica de Casos de Uso levaram à definição de um método de especificação de requisitos para a TV Digital Interativa que utiliza conceitos conhecidos pela comunidade de software. O método foi utilizado em um projeto real, e é abordado no artigo como relato de experiência. O trabalho completo pode ser encontrado no endereço abaixo.

Fonte: <http://revistas.ua.pt/index.php/prismacom/article/view/784>. Acesso em: 07 jun. 2012.

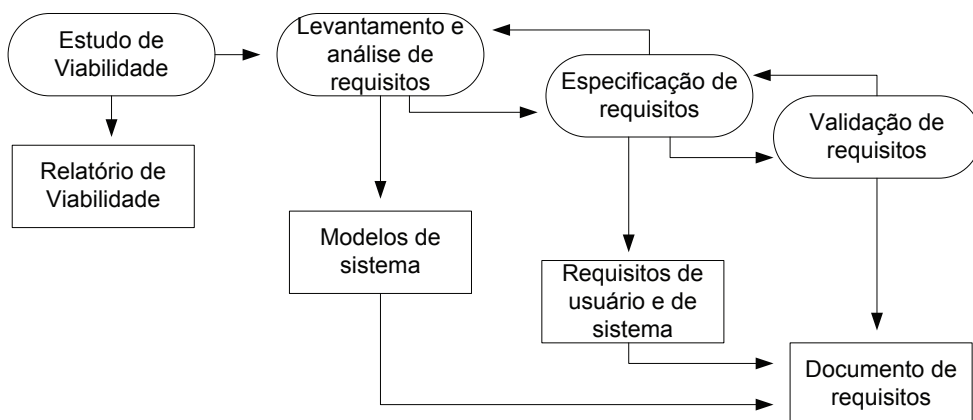
Anotações

ENGENHARIA DE REQUISITOS

Como foi dito na unidade anterior, a engenharia de requisitos é um processo que envolve todas as atividades necessárias para a criação e manutenção de um documento de requisitos de software. Existem quatro atividades genéricas de processo de engenharia de requisitos que são de alto nível: (i) o estudo de viabilidade do sistema, (ii) o levantamento e análise de requisitos, (iii) a especificação de requisitos e sua documentação e, finalmente, (iv) a validação desses requisitos. A seguir abordaremos todas as atividades, com exceção da especificação de requisitos, que já foi discutida nesta unidade. A figura abaixo ilustra a relação entre essas atividades e mostra também os documentos produzidos em cada estágio do processo de engenharia de requisitos, de acordo com Sommerville (2011, p. 24).

<http://www.youtube.com/watch?v=P4ixBvRF4NY&feature=related>.

O vídeo mostra uma entrevista com Sergio Ayres, consultor com vasta experiência em Gestão e Governança Corporativa, abordando a Engenharia de Requisitos.

Anotações

Fonte: Somerville (2011, p. 24).

As atividades de engenharia de requisitos, mostradas nessa figura, dizem respeito ao levantamento, à documentação e à verificação dos requisitos. Porém, é necessário deixar claro que, em praticamente em todos os sistemas, os requisitos se modificam; as pessoas interessadas desenvolvem melhor compreensão do que elas querem que o software faça; a organização compradora do sistema sofre modificações; e são feitas alterações no hardware, no software e no ambiente organizacional do sistema (SOMMERVILLE, 2007, p. 95).

ESTUDO DE VIABILIDADE

Segundo Sommerville (2007, p. 97), para todos os sistemas novos, o processo de engenharia de requisitos de sistemas deve se iniciar com um estudo de viabilidade ou elicitação de requisitos. O estudo de viabilidade inicia-se com uma descrição geral do sistema e de como ele será utilizado dentro de uma organização, sendo que o resultado desse estudo deve ser um relatório que recomenda se vale a pena ou não realizar o processo de engenharia de requisitos e, consequentemente, o processo de desenvolvimento de sistemas.

Um estudo de viabilidade é um estudo rápido, direcionado, que se destina a responder a

Anotações

algumas perguntas:

1. O sistema contribui para os objetivos gerais da organização?
2. O sistema pode ser implementado com a utilização de tecnologia atual dentro das restrições de custo e de prazo?
3. O sistema pode ser integrado com outros sistemas já em operação?

A questão sobre se o sistema contribui ou não para os objetivos da empresa é fundamental, pois se um sistema não for compatível com esses objetivos, ele não terá nenhum valor real para a mesma. Embora isso possa parecer óbvio, muitas organizações desenvolvem sistemas que não contribuem para seus objetivos, seja porque não existe uma declaração clara desses objetivos ou porque outros fatores políticos ou organizacionais influenciam na aquisição do sistema (SOMMERVILLE, 2007, p. 97).

Preparar um estudo de viabilidade envolve avaliar e coletar informações e redigir relatórios. A fase de avaliação identifica as informações exigidas para responder às três perguntas apresentadas anteriormente. Uma vez identificadas as informações, é preciso questionar as fontes de informação, a fim de encontrar as respostas para essas perguntas. Eis alguns exemplos das possíveis perguntas que devem ser feitas:

Como a organização se comportaria, se esse sistema não fosse implementado?

Quais são os problemas com os processos atuais e como um novo sistema ajudaria a diminuir esses problemas?

Que contribuição direta o sistema trará para os objetivos da empresa?

As informações podem ser transferidas para outros sistemas organizacionais e também podem ser recebidas a partir deles?

O sistema requer tecnologia que não tenha sido utilizada anteriormente na organização?

Anotações

O que precisa e o que não precisa ser compatível com o sistema?

Entre as fontes de informação estão os gerentes de departamentos em que o sistema será utilizado, os engenheiros de software que estão familiarizados com o tipo de sistema proposto, peritos em tecnologia, usuários finais de sistema entre outros. Eles devem ser entrevistados durante o estudo de viabilidade, a fim de coletar as informações exigidas.

O relatório do estudo de viabilidade deverá ser elaborado com base nas informações mencionadas acima, e deve recomendar se o desenvolvimento do sistema deve continuar ou não. Ele pode propor mudanças no enfoque, no orçamento e no cronograma, além de sugerir outros requisitos de alto nível para o sistema.

LEVANTAMENTO E ANÁLISE DE REQUISITOS

De acordo com Sommerville (2007, p. 97), após os estudos iniciais de viabilidade, a próxima atividade do processo de engenharia de requisitos é o levantamento e a análise de requisitos. Nessa atividade, os membros da equipe técnica de desenvolvimento de software trabalham com o cliente e os usuários finais do sistema para descobrir mais informações sobre o domínio da aplicação, que serviços o sistema deve fornecer, o desempenho exigido do sistema, as restrições de hardware e assim por diante.

O levantamento e a análise de requisitos podem envolver diferentes tipos de pessoas em uma organização. O termo *stakeholder* é utilizado para se referir a qualquer pessoa que terá alguma influência direta ou indireta sobre os requisitos do sistema. Dentre os *stakeholders* destacam-se os usuários finais que interagirão com o sistema e todo o pessoal, em uma organização, que venha a ser por ele afetado. Os engenheiros que estão desenvolvendo o sistema ou fazendo a manutenção de outros sistemas relacionados, os gerentes de negócios, os especialistas nesse domínio, os representantes de sindicato entre outros, podem ser também os *stakeholders* do sistema.

Anotações

O levantamento e a análise de requisitos compõem um processo difícil, por diversas razões (SOMMERVILLE, 2007, p. 98):

1. Os *stakeholders* frequentemente não sabem na realidade o que querem do sistema computacional, a não ser em termos muito gerais; eles podem achar difícil articular o que desejam do sistema, muitas vezes, fazendo pedidos não realistas, por não terem noção do custo de suas solicitações.
2. Os *stakeholders* em um sistema expressam naturalmente os requisitos em seus próprios termos e com o conhecimento implícito de sua área de atuação, dificultando a compreensão por parte dos engenheiros de software que não têm experiência no domínio do cliente.
3. Diferentes *stakeholders* têm em mente diferentes requisitos e podem expressá-los de maneiras distintas, obrigando os engenheiros de software a descobrir todas as possíveis fontes de requisitos e a encontrar os pontos comuns e os conflitos.
4. Fatores políticos podem influenciar os requisitos do sistema.
5. O ambiente econômico e de negócios, no qual a análise de requisitos ocorre, é dinâmico, mudando durante o processo de análise. Como consequência, a importância dos requisitos específicos pode mudar, podendo surgir novos requisitos por parte dos novos stakeholders, que não haviam sido consultados inicialmente.

Introdução à Engenharia de Requisitos

Por Rodrigo Oliveira Spínola, Doutor e Mestre em Engenharia de Sistemas e Computação. Fonte: <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-engenharia-de-requisitos/8034>. Acesso em: 13 jun. 2012.

Entrevista

As entrevistas formais ou informais com os *stakeholders* do sistema faz parte da maioria dos

processos de engenharia de requisitos. É a fonte mais importante para o levantamento dos requisitos desde que o entrevistado confie no entrevistador.

A sumarização durante e no final da entrevista é necessária primeiro para garantir que toda informação apresentada foi anotada e segundo que foi corretamente entendida.

Antes de tentar uma entrevista, o engenheiro de software deve prepará-la:

a) Comece por definir os objetivos. Verifique a documentação formal e desenvolva um esquema do sistema existente ou proposto. Identifique questões, partes omitidas e ambíguas. Estes fatos ou componentes desconhecidos representam um esboço inicial dos objetivos. Pode ser necessário entrevistar várias pessoas para atingir o objetivo.

b) Selecionar a pessoa ou grupo a ser entrevistado. É claro que você quer encontrar a pessoa que melhor possa responder sobre o assunto. Pode-se encontrá-la utilizando o organograma, uma análise do fluxo de trabalho ou uma lista de distribuição de relatórios. Comece pelo organograma e pelo gerente que parece ser o melhor para responder às questões. Além disso, as pessoas ficam menos hesitantes se souberem que a entrevista foi autorizada pelo chefe.

c) Ler a documentação relevante, conhecer a posição e as responsabilidades do entrevistado, ocupar-se com documentos ou procedimentos relevantes.

d) Preparar questões específicas. Selecione questões específicas que podem ser respondidas. Desenvolva uma lista de questões a serem seguidas se a entrevista começar a se desviar do ponto-chave.

A entrevista deve ser marcada com antecedência, o horário deve ser combinado e as questões devem ser preparadas.

Uma entrevista é composta de três partes: a abertura, o corpo e o fechamento.

ABERTURA - o objetivo-chave é estabelecer harmonia (concordância). Comece se identificando, apresentando o tópico que pretende discutir e o propósito (objetivo) da entrevista. Se houver necessidade, "quebre o gelo" com conversas informais, mas não caia na "perda de tempo".

CORPO - pode-se começar com uma questão relativamente aberta (Quando eu li a documentação para este sistema, tive algum trabalho com (anuncie a parte ou seção) você pode me explicar?) e gradualmente, caminhe através de questões específicas.

Nesta fase, o engenheiro de software deve:

a) Mostrar que conhece as responsabilidades e deveres do trabalho do entrevistado.

Exemplo: isto é o que eu entendo do seu trabalho (uma breve descrição) está correto?

b) Procurar saber as decisões que o entrevistado toma (quais são e como ele toma as decisões; quais são as informações necessárias, se da forma como são apresentadas são satisfatórias, qual o tempo necessário - antecedência - para que se possa tomar as decisões).

c) Procurar respostas quantitativas. Exemplo: quantos telefones, funcionário você tem no departamento?

d) Evitar falar palavras sem sentido (falar baixo, fazer generalizações, termos técnicos).

e) Ouvir as respostas. Dê tempo para o entrevistado responder, não saia com respostas antecipadas. Não se concentre na próxima questão (isto é um erro comum dos iniciantes). A lista de questões preparada é apenas um guia. Tenha certeza de que as questões são

Anotações

relevantes, evite questões complexas e desnecessárias.

f) Pedir explicações para as questões que ficarem obscuras.

g) Pedir ideias e sugestões e descobrir se o entrevistado quer que sejam consideradas. Exemplo: você tem alguma sugestão ou recomendações relativas ao método para calcular o orçamento? Você gostaria que os seus superiores ou os demais ficassem sabendo de suas sugestões?

ENCERRAMENTO - se a entrevista tiver consumido mais tempo do que o previsto peça para continuar e ofereça uma reprogramação. Quando tiver toda a informação necessária, agradeça e faça um sumário de todos os pontos principais. Avise se for necessária outra sessão de entrevista com a mesma pessoa.

Muitas vezes, algumas expressões corporais podem substituir ou comunicar mais informações do que as próprias palavras. Este tipo de comunicação pode ajudar o engenheiro de software a:

a) interpretar as palavras do entrevistado;

b) determinar a atitude geral do entrevistado para as questões que estão sendo discutidas;

c) avaliar a confidência que o entrevistado demonstrou tanto ao seu redor como no tratamento da área de abrangência do sistema.

Vários pontos devem ser aprendidos e esclarecidos na entrevista:

a) Organização da empresa (ambiente de trabalho). Como o administrador organiza o seu pessoal? Como esta organização se relaciona às funções maiores que a empresa executa?

b) Os objetivos e exigências do sistema (declarados nos manuais de procedimentos) devem ser reafirmados e esclarecidos na entrevista - muitas vezes os objetivos e exigências

Anotações

declarados nos manuais não são os mesmos que os representantes veem. Quando existe uma discrepância, é possível que as metas representadas nos documentos possam ser irreais com o atual potencial humano. O tempo e o crescimento podem ter alterado a meta declarada.

c) Fluxo funcional: para cada função importante, determinar as etapas exigidas e descreva o significado delas.

d) Exigência de recursos: determinar quais são os recursos aplicados pela organização para executar o trabalho. Quais são as exigências com:

→ Recursos humanos (treinamento especializado, experiência exigida).

→ Equipamento e material necessário para apoiar na execução do trabalho.

e) Relação de tempo: como o trabalho executado se relaciona a períodos específicos do ano ou outros ciclos comerciais. Existe pico? Qual o atual volume de trabalho?

f) Formulários, procedimentos e relatórios → quais são utilizados? (inclua exemplo de cada formulário, relatório e procedimento).

→ Verifique se o material tem origem no escritório, se é modificado pelo escritório e/ ou transmitido para outro escritório.

→ Faça comparações que determinam se é inutilizado, duplicado ou incompleto.

→ Verifique a satisfação dos usuários com esses documentos.

g) Funções desejáveis e não existentes: registre a opinião das pessoas sobre o sistema, como ele existe e como poderia ser. Atenção → opiniões mais subjetivas que objetivas.

h) Flexibilidade dos procedimentos: o sistema atual é tão rígido e inflexível que a menor modificação requer o maior remendo?

Anotações

i) Capacidade: o sistema atual consegue manipular volumes maiores do que aqueles que resultam do crescimento normal?

"Coloque três interessados em uma sala e pergunte a eles que tipo de sistema desejam. Provavelmente você obterá quatro ou mais opiniões diferentes".

Autor desconhecido.

ESPECIFICAÇÃO DE REQUISITOS

Durante o levantamento de requisitos (levantamento de dados), a equipe de desenvolvimento tenta entender o domínio (contexto/problema) que deve ser automatizado, sendo que o produto do levantamento de requisitos é o **DOCUMENTO DE REQUISITOS ou ESPECIFICAÇÃO DE REQUISITOS**, que declara os diversos tipos de requisitos do sistema (requisitos funcionais, requisitos não funcionais, de usuário e de sistema). Já tratamos desse tópico nesta unidade.

VALIDAÇÃO DE REQUISITOS

A validação de requisitos tem como objetivo mostrar que os requisitos realmente definem o sistema que o cliente deseja. Ela tem muito em comum com a análise de requisitos, uma vez que se preocupa em descobrir problemas nos requisitos. Contudo, esses são processos distintos, já que a validação deve se ocupar da elaboração de um esboço completo do documento de requisitos, enquanto a análise envolve trabalhar com requisitos incompletos (SOMMERVILLE, 2007, p. 105).

A validação de requisitos é importante porque a ocorrência de erros em um documento de requisitos pode levar a grandes custos relacionados ao retrabalho, quando esses erros são descobertos durante o desenvolvimento ou depois que o sistema estiver em operação. O custo de fazer uma alteração no sistema, resultante de um problema de requisito, é muito maior do que reparar erros de projeto e de codificação, pois, em geral, significa que o projeto do sistema e a implementação também devem ser modificados e que o sistema tem de ser novamente testado.

Durante a etapa de validação de requisitos, Sommerville (2007, p. 106) propõe que diferentes tipos de verificação devem ser realizados sobre os requisitos no documento de requisitos. Dentre as verificações destacam-se:

1. **Verificações de validade**. Um usuário pode considerar que um sistema é necessário para realizar certas funções. Contudo, mais estudos e análises podem identificar funções adicionais ou diferentes, que são exigidas. Os sistemas têm diversos usuários com necessidades diferentes e qualquer conjunto de requisitos é inevitavelmente uma solução conciliatória da comunidade de usuários.
2. **Verificações de consistência**. Os requisitos em um documento não devem ser conflitantes, ou seja, não devem existir restrições contraditórias ou descrições diferentes para uma mesma função do sistema.
3. **Verificações de completeza**. O documento de requisitos deve incluir requisitos que definam todas as funções e restrições exigidas pelos usuários do sistema.

Anotações

___

___

___

___

4. **Verificações de realismo**. Utilizando o conhecimento da tecnologia existente, os requisitos devem ser verificados, a fim de assegurar que eles realmente podem ser implementados, levando-se também em conta o orçamento e os prazos para o desenvolvimento do sistema.
5. **Facilidade de verificação**. Para diminuir as possíveis divergências entre cliente e fornecedor, os requisitos do sistema devem sempre ser escritos de modo que possam ser verificados. Isso implica na definição de um conjunto de verificações para mostrar que o sistema entregue cumpre com esses requisitos.

Algumas técnicas de validação de requisitos podem ser utilizadas em conjunto ou individualmente. A seguir são mostradas algumas delas.

1. **Revisões de requisitos**. Os requisitos são analisados sistematicamente por uma equipe de revisores, a fim de eliminar erros e inconsistências.
2. **Prototipação**. Nessa abordagem de validação, um modelo executável do sistema é mostrado aos usuários finais e clientes, possibilitando que os mesmos experimentem o modelo para verificar se atende às necessidades da empresa.
3. **Geração de casos de teste**. Como modelo ideal, os requisitos deveriam ser testáveis. Se os testes para os requisitos são criados como parte do processo de validação, isso, muitas vezes, revela problemas com os requisitos. Se um teste é difícil ou impossível de ser projetado, isso frequentemente significa que os requisitos serão de difícil implementação e devem ser reconsiderados. O desenvolvimento de testes a partir dos requisitos de usuário, antes da implementação do sistema, é uma parte integrante da *Extreme Programming*.

As dificuldades da validação de requisitos não devem ser subestimadas, pois é muito difícil demonstrar que, de fato, um conjunto de requisitos atende às necessidades de um usuário. Os usuários devem pensar no sistema em operação e imaginar como esse sistema se adequaria ao seu trabalho. Não é fácil para profissionais habilitados de computação conseguir realizar esse tipo de análise abstrata, imagina só como será difícil para os usuários de sistema. Sendo assim, a validação de requisitos não consegue descobrir todos os problemas com os requisitos, implicando em modificações para corrigir essas omissões e falhas de compreensão, depois que o documento de requisitos foi aceito (SOMMERVILLE, 2011, p. 77).

Anotações

"Gastamos um bom tempo – a maior parte do esforço de um projeto – não implementando ou testando, mas sim tentando decidir o que construir".

Brian Lawrence, ex-jogador de beisebol da Major League.

Uso de protótipos de interface como idioma durante a Validação de requisitos – Uma análise semiótica

Por Carlos Eduardo Marquioni, M.Sc., PMP

Fonte: <http://www.marquioni.com.br/artigos.php?id_artigo=14&titulo=Uso de protótipos de interface como idioma durante a Validação de requisitos – Uma análise semiótica>. Acesso em: 12 jun. 2012.

CONSIDERAÇÕES FINAIS

Caro(a) aluno(a), chegamos ao final da terceira unidade, na qual estudamos o assunto Requisitos de Software. Nesta unidade, foi mostrado que os requisitos para um software devem estabelecer o que o sistema deve fazer e definir também as restrições sobre o seu funcionamento e implementação.

Os requisitos de software podem ser classificados em requisitos funcionais – que são aqueles serviços que o sistema deve fornecer – e requisitos não funcionais – que estão, frequentemente, relacionados às propriedades emergentes do sistema, aplicando-se ao sistema como um todo.

Todos os requisitos, sejam eles funcionais ou não funcionais, devem ser definidos da forma mais clara possível para que não haja problemas na sua interpretação, pois é a partir da definição desses requisitos que o sistema será modelado, projetado, implementado, testado e, por fim, colocado em funcionamento. Um erro na definição de requisitos pode levar a alterações

em todas essas etapas, por isso essa etapa é tão importante.

Para auxiliar todo esse processo, vimos que Sommerville (2011) propõe que a engenharia de requisitos seja um processo que deve incluir quatro atividades de alto nível. A primeira atividade servirá para avaliar se o sistema será útil para a empresa. Isto se dá por meio do estudo de viabilidade, sendo que, ao final desta atividade, um relatório de viabilidade deve ser elaborado, no qual se recomenda se vale a pena ou não realizar o processo de engenharia de requisitos e, consequentemente, o processo de desenvolvimento de sistemas. Após o estudo de viabilidade, parte-se para a descoberta dos requisitos, ou seja, é realizado o levantamento e a análise dos requisitos, envolvendo diferentes tipos de pessoas (*stakeholders*) na organização. Existem diversas formas para que esse levantamento seja realizado, além da entrevista, que foi mostrada nesta unidade. A entrevista é a forma mais utilizada, por isso, optamos em descrevê-la.

A terceira atividade do processo de engenharia de requisitos é a conversão dos requisitos levantados em uma forma-padrão, ou seja, em um documento de requisitos de software. Por meio do exemplo mostrado, que foi um documento de requisitos para uma locadora de filmes, pode-se perceber que ele deve trazer detalhes suficientes para dar continuidade ao processo de desenvolvimento do sistema. E, finalmente, a última atividade, é a verificação de que os requisitos realmente definem o sistema que o cliente quer, ou seja, deve ser realizada a validação dos requisitos anotados no documento de requisitos. Note que, nem sempre é fácil validar os requisitos, pois, às vezes, o próprio cliente não sabe definir com precisão o que ele deseja, o que acaba dificultando essa etapa.

Na próxima unidade vamos conhecer como se dá o processo de modelagem de um sistema e vamos usar os conceitos de orientação a objetos apresentados na primeira unidade, pois a UML – *Unified Modeling Language* (Linguagem de Modelagem Unificada), que utilizaremos para realizar a modelagem, é baseada no paradigma da orientação a objetos. Nesta próxima unidade trabalharemos com uma ferramenta CASE para nos auxiliar na elaboração de alguns diagramas e você verá que o documento de requisitos é realmente muito importante. Vamos lá?

ATIVIDADE DE AUTOESTUDO

1. Identifique e descreva os quatro tipos de requisitos que podem ser definidos para um sistema.
2. Descreve três tipos de requisitos não funcionais que podem ser definidos para um sistema, fornecendo exemplos para cada um deles.
3. Baseando-se no Documento de Requisitos da Locadora de Filmes, mostrado como exemplo nesta unidade, relacione os requisitos funcionais encontrados no mesmo.
4. Descreva detalhadamente as quatro atividades que fazem parte do processo de engenharia de requisitos.

UNIDADE IV

MODELAGEM DE SISTEMAS

Professora Me. Márcia Cristina Dadalto Pascutti

Objetivos de Aprendizagem

- Expor a importância da modelagem de sistemas.
- Mostrar os conceitos relacionados a diagramas de casos de uso e diagramas de classes.
- Mostrar um estudo de caso no qual são construídos os diagramas de casos de uso e de classes.

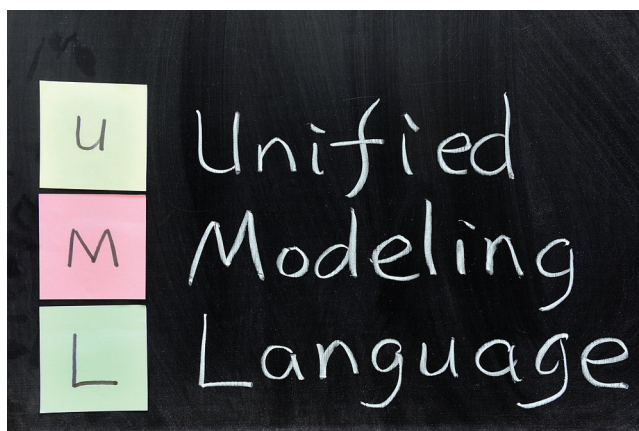
Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- **Modelagem de Sistemas**
- **Diagrama de Casos de Uso**
- **Diagrama de Classes**

INTRODUÇÃO

Caro(a) aluno(a), na terceira unidade estudamos sobre os requisitos de um sistema e foi bastante destacada a importância de um documento de requisitos. Nesta unidade você verá que, a partir do documento de requisitos, realizaremos a modelagem de um sistema.

A modelagem de sistema é o processo de elaboração de modelos abstratos de um sistema, normalmente representado por meio de um diagrama, em que cada um desses modelos apresenta uma visão ou perspectiva diferente do sistema (SOMMERVILLE, 2011 p. 82). Esses modelos, normalmente, são elaborados utilizando-se uma notação gráfica, que, em nosso caso, será a UML.

De acordo com BOOCH (2005, p. 13), "a UML é uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software".

Da mesma forma que os arquitetos elaboram plantas e projetos para serem usados, para a construção de um edifício, os engenheiros de software criam os diagramas UML para auxiliar os desenvolvedores de software a construir o software.

Anotações

A UML foi desenvolvida, inicialmente, por meio da combinação de um grupo de notações de modelagem usadas pela indústria de software: o método de Booch, o método OMT (*Object Modeling Technique*) de Jacobson e o método OOSE (*Object-Oriented Software Engineering*) de Rumbaugh. Em 1997, a UML 1.0 foi oferecida para padronização ao OMG (*Object Management Group*). A UML 1.0 foi revisada com a ajuda de vários segmentos da comunidade de desenvolvedores de software, tornando-se UML 1.1, sendo adotada pelo OML em novembro de 1997. O padrão atual é a UML 2.0, sendo um padrão ISO.

A UML define em sua versão 2.0 treze tipos de diferentes diagramas para uso na modelagem de software. Nesta unidade, veremos somente os diagramas de casos de uso e de classe. Se você quiser conhecer os outros diagramas, consulte alguns livros relacionados nas referências. Como nosso objetivo aqui é mostrar a modelagem de um sistema, utilizaremos somente esses dois diagramas.

Primeiramente, será apresentado a você, caro(a) aluno(a), o diagrama de casos de uso. Esse diagrama ajuda a determinar a funcionalidade e as características do sistema sob o ponto de vista do usuário, sendo um dos diagramas mais gerais e informais da UML. Para a elaboração do diagrama de casos de uso, deve ser utilizada uma linguagem simples e de fácil compreensão para que os usuários possam ter uma ideia geral de como o sistema irá se comportar (GUEDES, 20007, p. 15).

Depois, será mostrado o diagrama de classes. Esse é o diagrama mais importante e também o mais utilizado da UML, e serve de apoio para a maioria dos outros diagramas (que não serão abordados neste livro). O diagrama de classes define a estrutura das classes identificadas para o sistema, mostrando os atributos e métodos possuídos por cada uma das classes, além de estabelecer como as classes se relacionam e trocam informações entre si.

Anotações

MODELAGEM DE SISTEMAS

A necessidade de planejamento no desenvolvimento de sistemas de informação leva ao conceito de modelagem de software, ou seja, antes do software ser concebido deve-se criar um modelo para o mesmo. Um modelo pode ser visto como uma representação idealizada de um sistema a ser construído. Exemplos de modelos: maquetes de edifício, plantas de casa, fluxogramas etc.

A modelagem de sistemas de software consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema. São várias as razões para se utilizar modelos na construção de sistemas, eis algumas:

- No desenvolvimento de software usamos desenhos gráficos denominados de diagramas para representar o comportamento do sistema. Esses diagramas seguem um padrão lógico e possuem uma série de elementos gráficos que possuem um significado pré-definido.
- Apesar de um diagrama conseguir expressar diversas informações de forma gráfica, em diversos momentos há a necessidade de adicionar informações na forma de texto, com o objetivo de explicar ou definir certas partes desse diagrama. A modelagem de um sistema em forma de diagrama, juntamente com a informação textual associada, formam a documentação de um sistema de software.
- O rápido crescimento da capacidade computacional das máquinas resultou na demanda por sistemas de software cada vez mais complexos, que tirassem proveito de tal capacidade. Por sua vez, o surgimento desses sistemas mais complexos resultou na necessidade de reavaliação da forma de desenvolver sistemas. Desde o aparecimento do primeiro computador até os dias de hoje, as técnicas para construção de sistemas computacionais têm evoluído para suprir as necessidades do desenvolvimento de software.
  - Década de 1950/60: os sistemas de software eram bastante simples e dessa forma as técnicas de modelagem também. Era a época dos fluxogramas e diagramas de módulos.
  - Década de 1970: nessa época houve uma grande expansão do mercado computacional. Sistemas complexos começavam a surgir e, por consequência, modelos

Anotações

mais robustos foram propostos. Nesse período, surge a programação estruturada e no final da década a análise e o projeto estruturado.

  - Década de 1980: surge a necessidade por interfaces homem-máquina mais sofisticadas, o que originou a produção de sistemas de software mais complexos. A análise estruturada se consolidou na primeira metade dessa década e em 1989 Edward Yourdon lança o livro Análise Estruturada Moderna, tornando-o uma referência no assunto.
  - Década de 1990: nesse período surge um novo paradigma de modelagem, a Análise Orientada a Objetos, como resposta a dificuldades encontradas na aplicação da Análise Estruturada a certos domínios de aplicação.
  - Final da década de 90 e momento atual: o paradigma da orientação a objetos atinge a sua maturidade. Os conceitos de padrões de projetos (*design patterns*), *frameworks* de desenvolvimento, componentes e padrões de qualidade começam a ganhar espaço. Nesse período, surge a Linguagem de Modelagem Unificada (UML), que é a ferramenta de modelagem utilizada no desenvolvimento atual de sistemas.

O processo de desenvolvimento de software é uma atividade bastante complexa. Isso se reflete no alto número de projetos de software que não chegam ao fim, ou que extrapolam recursos de tempo e de dinheiro alocados. Em um estudo clássico sobre projetos de desenvolvimento de software realizado em 1994 foi constatado que:

- Porcentagem de projetos que terminam dentro do prazo estimado: 10%.
- Porcentagem de projetos que são descontinuados antes de chegarem ao fim: 25%.
- Porcentagem de projetos acima do custo esperado: 60%.
- Atraso médio nos projetos: um ano.

Os modelos construídos na fase de análise devem ser cuidadosamente validados e verificados. O objetivo da validação é assegurar que as necessidades do cliente estão sendo atendidas. Nessa atividade, os analistas apresentam os modelos criados para representar o sistema aos futuros usuários para que esses modelos sejam validados.

Anotações

A verificação tem o objetivo de analisar se os modelos construídos estão em conformidade com os requisitos definidos. Na verificação dos modelos, são analisadas a exatidão de cada modelo em separado e a consistência entre os modelos. A verificação é uma etapa típica da fase de projeto que é a próxima etapa do desenvolvimento de software.

Caro(a) aluno(a), utilizaremos a UML para realizar a modelagem de sistemas, e na primeira unidade estudamos alguns conceitos relacionados à orientação a objetos e também fizemos uma introdução à linguagem UML. Lembre-se que os artefatos de software produzidos durante a modelagem servirão de base para a fase seguinte do ciclo de desenvolvimento de sistemas, ou seja, a fase projeto.

DIAGRAMA DE CASOS DE USO

O diagrama de casos de uso (*Use Case Diagram*) é, dentre todos os diagramas da UML, o mais abstrato, flexível e informal, sendo utilizado principalmente no início da modelagem do sistema, a partir do documento de requisitos, podendo ser consultado e possivelmente modificado durante todo o processo de engenharia e também serve de base para a modelagem de outros diagramas (GUEDES, 2007, p. 38).

O principal objetivo deste diagrama é modelar as funcionalidades e serviços oferecidos pelo sistema, buscando, por meio de uma linguagem simples, demonstrar o comportamento externo do sistema da perspectiva do usuário.

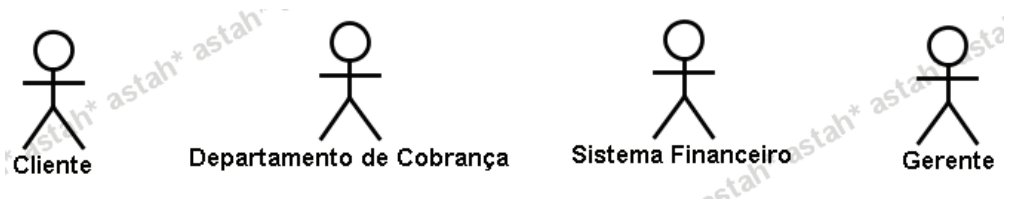
De acordo com Silva (2007, p. 101), o diagrama de caso de uso incorpora o conjunto de requisitos funcionais estabelecidos para o software que está sendo modelado. Esses requisitos devem estar descritos no documento de requisitos, como já explicamos na unidade anterior. Há uma correspondência entre os requisitos funcionais previstos para o software e os casos de uso. Os requisitos não funcionais não aparecem no diagrama de casos de uso, pois não constituem o foco da modelagem que estamos realizando.

Anotações

O diagrama de casos de uso é composto por atores, casos de uso e seus relacionamentos. A seguir descreveremos cada um desses elementos.

Atores

Um ator representa um papel que um ser humano, um dispositivo de hardware ou até outro sistema desempenha com o sistema (BOOCH, 2005, p. 231). Assim, um ator pode ser qualquer elemento externo que interaja com o software, sendo que o nome do ator identifica qual é o papel assumido por ele dentro do diagrama (GUEDES, 2007, p. 38).

Um caso de uso é sempre iniciado por um estímulo de um ator; ocasionalmente, outros atores podem participar do caso de uso. A figura abaixo apresenta alguns exemplos de atores.

Exemplos de Atores

Casos de Uso

De acordo com Booch (2005, p. 227), um caso de uso especifica o comportamento de um sistema ou de parte de um sistema, referindo-se a serviços, tarefas ou funções apresentadas pelo sistema, como cadastrar funcionário ou emitir relatório de produtos.

No diagrama de caso de uso não é possível documentar os casos de uso e nem a UML oferece um recurso para que isso seja feito, porém, é indicado que cada caso de uso seja documentado, demonstrando qual o comportamento pretendido para o caso de uso em questão e quais funções ele executará quando for solicitado. Essa documentação deverá, também, ser elaborada de acordo com o documento de requisitos e poderá auxiliar o desenvolvedor na

Anotações

elaboração dos demais diagramas da UML. A figura abaixo apresenta alguns exemplos de casos de uso.

Exemplos de Casos de Uso

Normalmente, os nomes de casos de uso são breves expressões verbais ativas, nomeando algum comportamento encontrado no vocabulário do sistema cuja modelagem está sendo realizada, a partir do documento de requisitos (BOOCH, 2005, p. 231). Alguns verbos que podem ser usados para nomear os casos de uso: efetuar, cadastrar, consultar, emitir, registrar, realizar, manter, verificar entre outros.

Relacionamento entre Casos de Uso e Atores

Conforme Melo (2004, p. 60), os casos de uso representam conjuntos bem definidos de funcionalidades do sistema, precisando relacionar-se com outros casos de uso e com atores que enviarão e receberão mensagens destes. Podemos ter os relacionamentos de associação, generalização, extensão e inclusão, da seguinte forma:

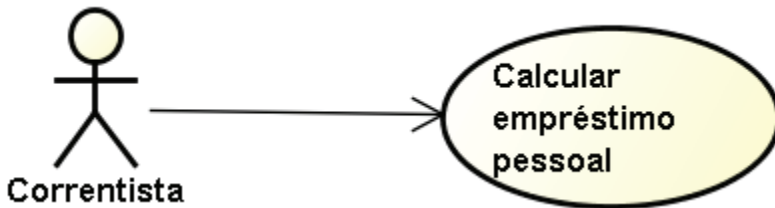
- Para relacionamentos entre **atores** e **casos de uso à** → somente **associação**;
- Para relacionamentos de **atores**, **entre si** → **à** somente **generalização**;
- Para relacionamentos de **casos de uso**, **entre si** → **à generalização**, **extensão** e **inclusão**.

Associação

A associação é o único relacionamento possível entre ator e caso de uso, sendo sempre binária, ou seja, sempre envolvendo apenas dois elementos. Melo (2004, p. 60) diz que "Representa a

Anotações

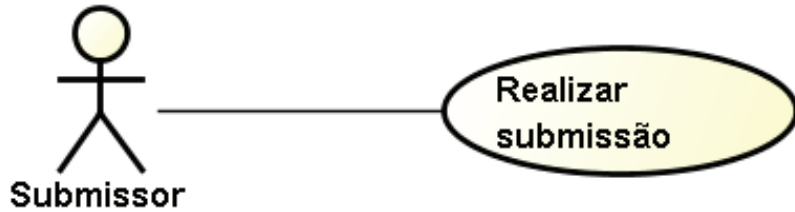
interação do ator com o caso de uso, ou seja, a comunicação entre atores e casos de uso, por meio do envio e recebimento de mensagens".

Um relacionamento de associação demonstra que o ator utiliza-se da funcionalidade representada pelo caso de uso. Esse tipo de relacionamento é representado por uma reta ligando o ator ao caso de uso. Pode ser que essa reta possua em sua extremidade uma seta, que indica a navegabilidade dessa associação, ou seja, se as informações são fornecidas pelo ator ao caso de uso (nesse caso a seta aponta para o caso de uso), se são transmitidas pelo caso de uso ao ator (nesse caso a seta aponta para o ator) ou ambos (neste último caso a reta não possui setas) (GUEDES, 2007, p. 40). A figura abaixo representa uma associação entre um ator e um caso de uso, em que o ator fornece uma informação para o caso de uso.

Associação entre um ator e um caso de uso

Veja o exemplo abaixo. Nele está sendo mostrado que o ator Gerente Administrativo recebe o Relatório de Vendas por Período (note que ele não solicita a emissão do relatório, ele somente recebe o relatório).

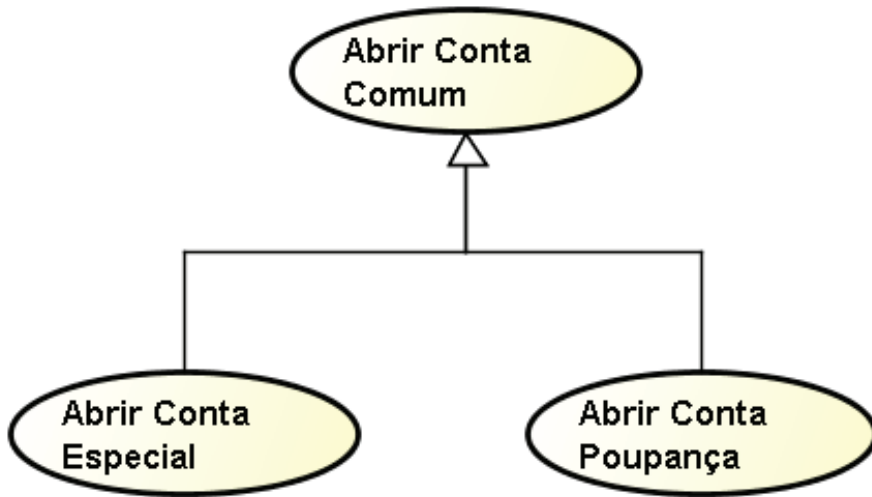
Associação entre um ator e um caso de uso

Anotações

Neste outro exemplo, percebemos que o ator denominado Submissor utiliza, de alguma forma, o serviço de Realizar Submissão e que a informação referente a esse processo trafega nas duas direções.

Associação entre um ator e um caso de uso

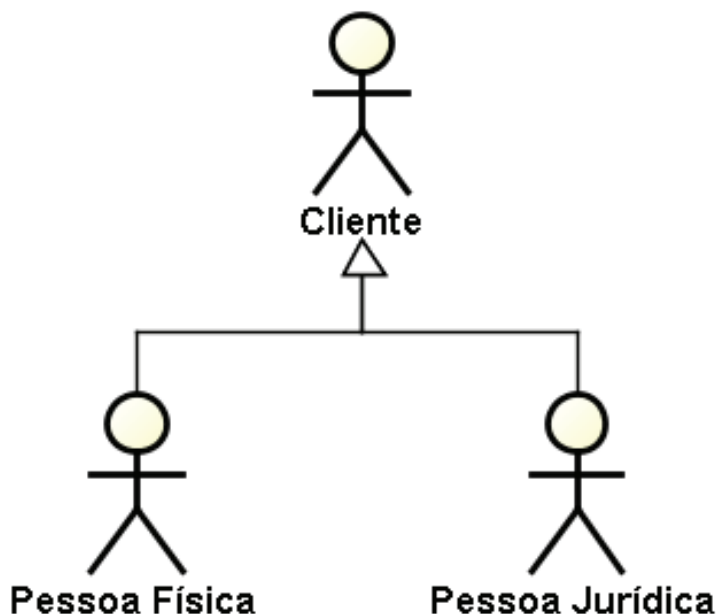
Generalização

Já explicamos o conceito de generalização/especialização quando falamos sobre herança. Aqui, no diagrama de casos de uso, também aplicamos esse conceito, ou seja, o relacionamento de generalização entre casos de uso pode ocorrer quando existirem dois ou mais casos de usos com características semelhantes, apresentando pequenas diferenças entre si. Quando isso acontece, define-se um caso de uso geral, que deverá possuir as características compartilhadas por todos os casos de uso em questão, e então relacionamos esse caso de uso geral com os casos de uso específicos, que devem conter somente a documentação das características específicas de cada um deles. Dessa forma, evita-se reescrever toda a documentação para todos os casos de uso envolvidos, porque toda a estrutura de um caso de uso generalizado é herdada pelos casos de uso especializados, incluindo quaisquer possíveis associações que o caso de uso generalizado possua. A associação é representada por uma reta com uma seta mais grossa, partindo dos casos de uso especializados e atingindo o caso de uso geral, como mostra a figura abaixo (GUEDES, 2007, p. 40).

Anotações

Exemplo de generalização entre casos de uso

Agora vamos entender o que este exemplo está representando: em um banco, existem três opções de abertura de contas: abertura de conta comum, de conta especial e de conta poupança, cada uma representada por um caso de uso diferente. As aberturas de conta especial e de conta poupança possuem todas as características e requisitos da abertura de conta comum, porém, cada uma delas possui também algumas características e requisitos próprios, o que justifica colocá-las como especializações do caso de uso Abertura de Conta Comum, detalhando-se as particularidades de cada caso de uso especializado em sua própria documentação (GUEDES, 2007, p. 41).

O relacionamento de generalização/especialização também pode ser aplicado entre atores. A figura abaixo apresenta um exemplo, em que existe um ator geral chamado Cliente e dois atores especializados chamados Pessoa Física e Pessoa Jurídica.

Anotações

Generalização entre Atores

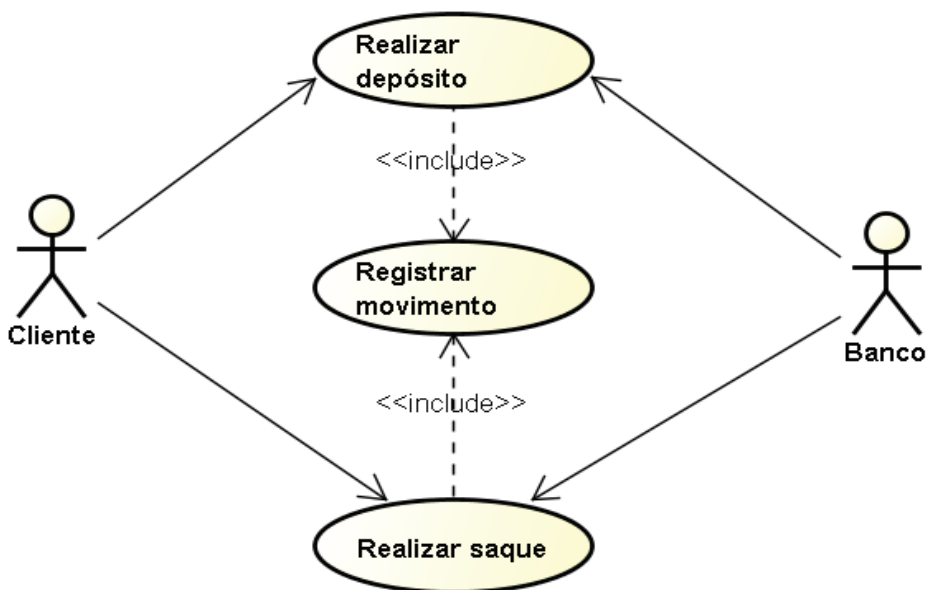
Inclusão

Este tipo de relacionamento é possível somente entre casos de uso e é utilizado quando existem ações comuns a mais de um caso de uso. Quando isso ocorre, a documentação dessas ações é colocada em um caso de uso específico, permitindo que outros casos de uso se utilizem dessas ações, evitando-se descrever uma mesma sequência de passos em vários casos de uso (GUEDES, 2007, p. 42).

Um relacionamento de inclusão entre casos de uso significa que o caso de uso base incorpora explicitamente o comportamento de outro caso de uso. O caso de uso incluído nunca permanece isolado, mas é apenas instanciado como parte de alguma base maior que o inclui (BOOCH, 2005, p. 235).

Anotações

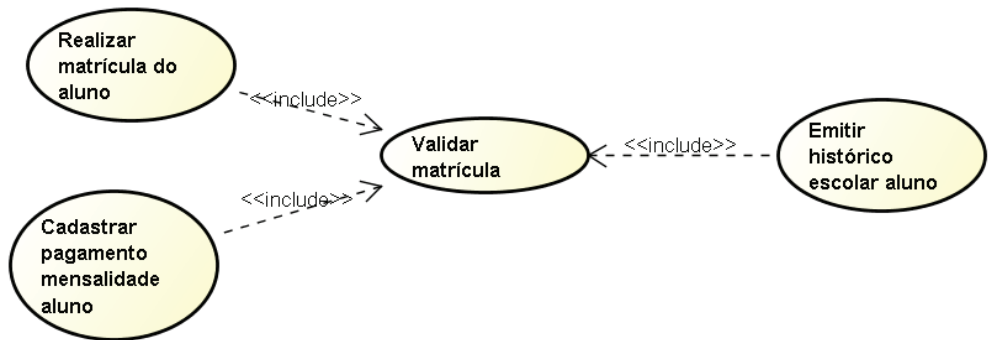
O relacionamento de inclusão pode ser comparado à chamada de uma sub-rotina, portanto, indicam uma obrigatoriedade, ou seja, quando um caso de uso base possui um relacionamento de inclusão com outro caso de uso, a execução do primeiro obriga também a execução do segundo (GUEDES, 2007, p. 42).

A representação do relacionamento de inclusão é feita por meio de uma reta tracejada contendo uma seta em uma de suas extremidades, e rotulada com a palavra-chave **<<include>>**. A seta deve sempre apontar para o caso de uso a ser incluído. Veja um exemplo na figura abaixo, que foi retirado de Guedes (2007, p. 43).

Exemplo de inclusão de caso de uso

Neste exemplo sempre que um saque ou depósito ocorrer, ele deve ser registrado para fins de histórico bancário. Como as rotinas para registro de um saque ou um depósito são extremamente semelhantes, colocou-se a rotina de registro em um caso de uso à parte,

Anotações

chamado Registrar Movimento, que será executado obrigatoriamente sempre que os casos de uso Realizar Depósito ou Realizar Saque forem utilizados. Assim, **só é preciso descrever os passos para registrar um movimento no caso de uso incluído (GUEDES, 2007, p. 43). Neste exemplo temos dois casos de uso base e um caso de a ser incluído.**

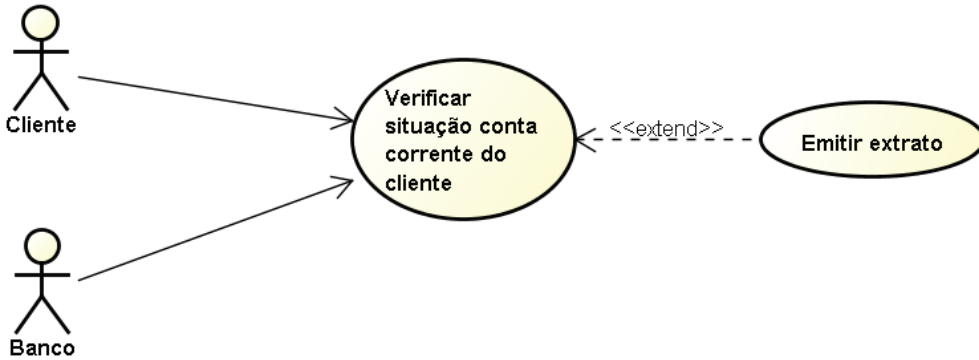
Agora veja outro exemplo na figura abaixo. Aqui, está sendo apresentada a seguinte situação: em algum ponto dos casos de uso Realizar matrícula do aluno (caso de uso base), Cadastrar pagamento mensalidade aluno (caso de uso base) e Emitir histórico escolar aluno (caso de uso base) é necessário Validar a matrícula (caso de uso a ser incluído) do aluno. Assim, nestes pontos o caso de uso Validar matrícula será internamente copiado.

Outro exemplo de inclusão de caso de uso

Extensão

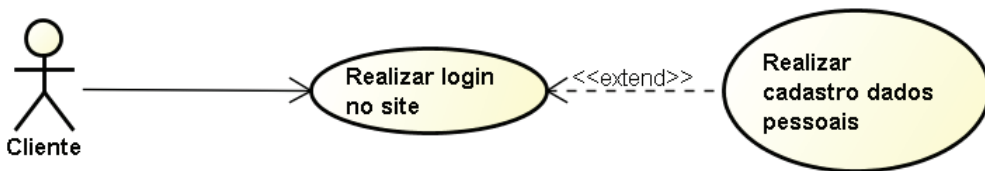
O relacionamento de extensão também é possível somente entre casos de uso e é utilizado para modelar rotinas opcionais de um sistema, que ocorrerão somente se uma determinada condição for satisfeita. A extensão separa um comportamento obrigatório de outro opcional.

As associações de extensão possuem uma representação muito semelhante às associações de inclusão, sendo também representadas por uma reta tracejada, diferenciando-se por possuir um estereótipo contendo o texto “<<extend>>” e pela seta apontar para o caso de uso

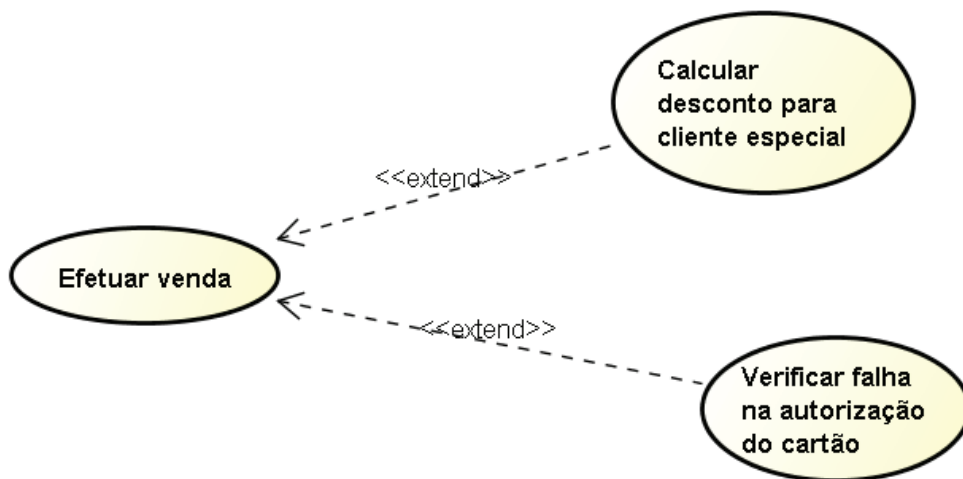
Anotações

que estende, ou seja, neste caso a seta aponta para o caso de uso base (GUEDES, 2007, p. 43). Veja abaixo um exemplo de relacionamento de extensão.

Exemplo de extensão de caso de uso

Neste exemplo, está sendo mostrado que tanto o Cliente quanto o Banco podem Verificar a situação da conta corrente do cliente e, se for o caso, pode-se emitir um extrato. Mas, note que, o extrato somente será emitido se alguma condição do caso de uso base – Verificar situação conta corrente do cliente for satisfeita, caso contrário, o extrato nunca será emitido.

Agora vamos mostrar um exemplo bastante comum que acontece quando utilizamos sistemas via Internet, em que, para utilizar os serviços, o cliente deve se logar no sistema e, caso seja a primeira vez, deverá realizar o seu cadastro pessoal.

Outro exemplo de extensão de caso de uso

Anotações

No exemplo acima, o caso de base é o Realizar login no site e Realizar cadastro dados pessoais é o caso de uso a ser estendido.

Representação de um Relacionamento Extend

QUADRO RESUMO DE RELACIONAMENTOS ENTRE CASOS DE USO E ATORES

	Associação	Especialização/ Generalização	Inclusão	Extensão
Caso de Uso e Caso de Uso	-----	OK	OK	OK
Ator e Ator	-----	OK	-----	-----
Ator e Caso de Uso	OK	-----	-----	-----

Anotações

<http://www.youtube.com/watch?v=hfN6n5fJfLc&feature=relmfu>.

Vídeo que mostra a importância da modelagem de sistemas, bem como trata da elaboração do diagrama de casos de uso.

ESTUDO DE CASO

Vamos mostrar um documento de requisitos de uma fábrica de colchões e depois o diagrama de casos de uso que foi elaborado com base neste documento. A ferramenta que utilizaremos para modelar o diagrama será o Astah, como já foi dito anteriormente. Outra observação importante é que esse documento de requisitos está bastante simplificado, seu principal objetivo é mostrar como os conceitos mostrados podem ser aplicados em um diagrama de casos de uso.

DOCUMENTO DE REQUISITOS – FÁBRICA DE COLCHÕES

A fábrica de colchões Mimindo Ltda compra matéria-prima, fabrica os colchões (casal ou solteiro) e vende para seus clientes de várias partes do país. Ela deseja gerenciar todos os processos com a utilização de um sistema que você vai desenvolver.

Mas, o sistema será desenvolvido em partes sendo o que a empresa precisa de imediato é lançar no sistema as **compras de matérias-primas (MP)** e **cadastrar os produtos acabados (PA)**.

Anotações

Algumas informações importantes que ajudará o entendimento do sistema

- Quando a Mimindo Ltda comprar MP (matéria-prima) de um fornecedor receberá uma nota de compra que deve ser lançada no sistema. Nesta nota constam as seguintes informações: o número da nota, data emissão, nome do fornecedor, CFOP (código fiscal de operação) e as MPs com as quantidades compradas de cada uma delas, bem como o seu valor unitário. Uma nota de compra pode conter N matérias-primas. No lançamento da nota de compra o sistema deve atualizar o estoque de MPs automaticamente.
- Essas MPs devem ser cadastradas separadas por grupos, sendo que uma MP poderá pertencer a um único grupo (Ex.: tecidos, espumas etc.).
- Os PAs (produtos acabados) são os colchões fabricados que a Mimindo Ltda irá vender (***esse sistema que você está desenvolvendo agora não tratará da venda***). Para fabricar um PA é necessário saber quais as matérias-primas que compõem esse PA, bem como a quantidade de cada matéria-prima que é utilizada na fabricação do PA. O sistema deve permitir o cadastro dos PAs e também as matérias-primas e as quantidades de cada MP usadas em cada um deles.
- Os cadastros e a movimentação do sistema deverão ser realizadas pelo departamento de compras.
- O sistema deve emitir os relatórios que estão definidos abaixo. A informação de quem solicita cada relatório bem como de quem recebe cada relatório está junto com a definição do relatório.

O sistema deve emitir os seguintes relatórios:

A) Lista de Fornecedores contendo o código, nome, cidade e fone. Este relatório será solicitado e recebido pelo Departamento de Compras.

B) Lista de MPs por grupo contendo o nome do grupo e para cada grupo os dados da matéria-prima: código, descrição, quantidade em estoque e valor unitário. Este relatório será solicitado pelo Departamento de Compras e recebido pelo Departamento de Compras e pelo Gerente de Compras.

C) Relação de Compras por Fornecedor, contendo Número da Nota, Nome do fornecedor, Data da emissão e Total da nota. Este relatório será solicitado e recebido pelo Departamento de Compras.

D) Lista de Produtos Acabados contendo o código, descrição, valor de venda e o tipo

Anotações

do colchão (se é solteiro, casal, berço, queen size ou king size). Para cada produto acabado, emitir a lista de matérias-primas utilizadas para fabricação do mesmo. Para cada matéria-prima, emitir o seu código, sua descrição e a quantidade utilizada na fabricação do PA. Este relatório será solicitado e recebido pelo Departamento de Compras e também será solicitado e recebido pelo Gerente de Compras.

A figura abaixo mostra uma possível solução para o problema que acabamos de descrever.

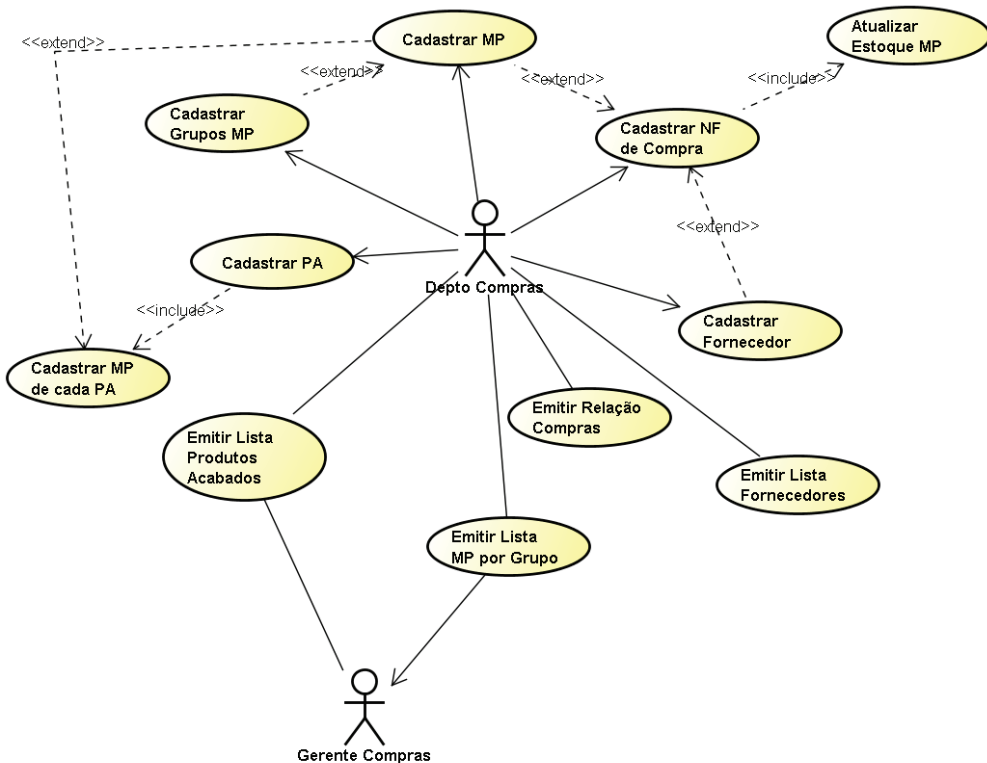



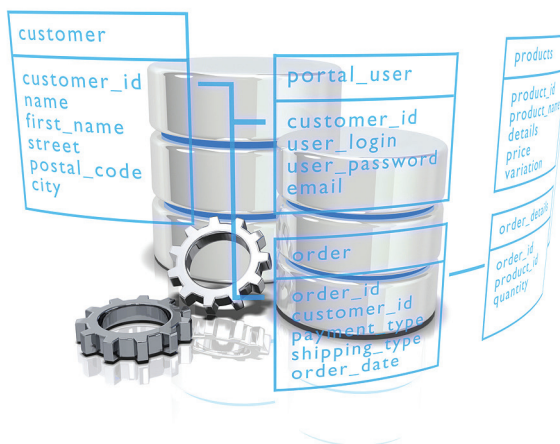
Diagrama de Casos de Uso para o Sistema Fábrica de Colchões

Anotações

DIAGRAMA DE CLASSES

O diagrama de classes tem como objetivo permitir a visualização das classes utilizadas pelo sistema e como essas se relacionam, apresentando uma visão estática de como essas classes estão organizadas, preocupando-se apenas em definir sua estrutura lógica (GUEDES, 2007, p. 52).

Ainda conforme Guedes (2007, p. 52), um diagrama de classes pode ser utilizado para modelar o modelo lógico de um banco de dados, parecendo-se, neste caso, com o Diagrama de Entidade-Relacionamento (Modelo Entidade-Relacionamento, que você estudará na disciplina de Banco de Dados). No entanto, deve ficar bem claro que o diagrama de classes não é utilizado unicamente para essa finalidade e mais importante, que uma classe não, necessariamente, corresponde a uma tabela.

Uma classe pode representar o repositório lógico dos atributos de uma tabela, porém, a classe não é a tabela, uma vez que os atributos de seus objetos são armazenados em memória enquanto uma tabela armazena seus registros fisicamente em disco. Além disso, uma classe possui métodos, que não existem em uma tabela.

Anotações

RELACIONAMENTOS

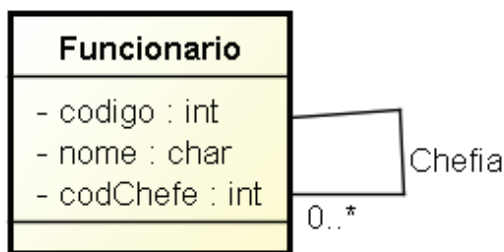
As classes não trabalham sozinhas, pelo contrário, elas colaboram umas com as outras por meio de relacionamentos (MELO, 2004, p. 108).

No diagrama de classes temos os relacionamentos de associação (que pode ser unária ou binária), generalização e agregação. Existem alguns tipos especiais de relacionamentos, porém não serão explicados aqui. Com os relacionamentos citados é possível elaborar um diagrama de classes.

Associação

De acordo com Melo (2004, p. 109), a associação é um relacionamento que conecta duas ou mais classes, demostrando a colaboração entre as instâncias de classe. Pode-se também ter um relacionamento de uma classe com ela mesma, sendo que, neste caso, tem-se a associação unária ou reflexiva.

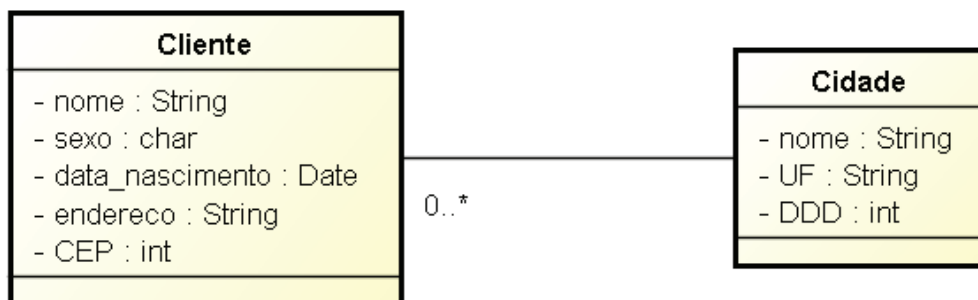
Associação Unária ou Reflexiva

Segundo Gedes (2007, p. 54), a associação unária ocorre quando existe um relacionamento de uma instância de uma classe com instâncias da mesma classe, conforme mostra o exemplo abaixo.

Exemplo de Associação Unária

Anotações

Neste exemplo, temos a classe Funcionário, cujos atributos são código, nome e o código do possível chefe do funcionário. Como esse chefe também é um funcionário, a associação chamada "Chefia" indica uma possível relação entre uma ou mais instâncias da classe Funcionário com outras instâncias da mesma classe, ou seja, tal associação determina que um funcionário pode ou não chefiar outros funcionários. Essa associação faz com que a classe possua o atributo codChefe para armazenar o código do funcionário que é responsável pela instância do funcionário em questão. Desse modo, após consultar uma instância da classe funcionário, pode-se utilizar o atributo codChefe da instância consultada para pesquisar por outra instância da Classe (GUEDES, 2007, p. 54).

Associação Binária

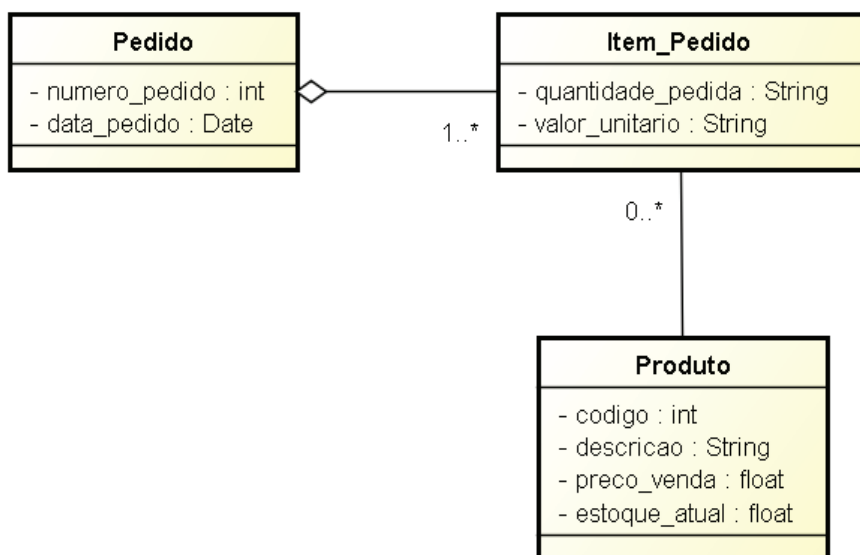
A associação binária conecta duas classes por meio de uma reta que liga uma classe a outra. A figura abaixo demonstra um exemplo de associação binária.

Exemplo de associação binária

Neste exemplo, um objeto da classe Cidade pode se relacionar ou não com instâncias da classe Cliente, conforme demonstra a multiplicidade 0..*, ao passo que, se existir um objeto da classe Cliente, ele terá que se relacionar com um objeto da classe Cidade, pois como não foi definida a multiplicidade na extremidade da classe Cidade, significa que ela é 1..1. Logo depois da explicação sobre os relacionamentos em um diagrama de classes, explicaremos o conceito de multiplicidade.

Anotações

Agregação

A agregação corresponde a um tipo especial de associação utilizada para expressar um relacionamento todo-parte. Ou seja, as informações do objeto-todo precisam ser complementadas pelas informações contidas em um ou mais objetos de outra classe, chamados objeto-parte, sendo que, os objetos-parte não podem ser destruídos por um objeto diferente do objeto-todo (GUEDES, 2007, p. 57).

O símbolo de agregação difere do de associação por conter um losango na extremidade da classe que contém os objetos-todo. A figura abaixo demonstra um exemplo de agregação, em que existe uma classe Pedido, que armazena os objetos-todo e uma classe Item_Pedido, em que são armazenados os objetos-parte.

Exemplo de agregação

Anotações

Neste exemplo, um pedido pode possuir apenas um item como vários, não sendo possível determinar o número máximo de itens que um pedido pode ter. Por isso, as informações da classe Pedido estão incompletas, possuindo apenas atributos que não se repetem. Os atributos que podem se repetir devem ser armazenados em uma classe dependente da classe Pedido. Dessa forma, sempre que uma instância da classe Pedido for pesquisada, todas as instâncias da classe Item_Pedido relacionadas à instância da classe Pedido pesquisada deverão também ser apresentadas. Da mesma forma, sempre que um objeto da classe Pedido for destruído, todos os objetos da classe Item_Pedido a ele relacionados devem também ser destruídos (GUEDES, 2007, p. 58).

Note que o relacionamento entre a classe Item_Pedido e Produto é de associação, portanto, quando o objeto da classe Item_pedido for excluído, por exemplo, o objeto relacionado a ele, na classe Produto, não deverá ser excluído.

Generalização/Especialização

Esta associação identifica classes-mãe (superclasse), chamadas gerais e classes-filhas (subclasses), chamadas especializadas, demonstrando a ocorrência de herança e possivelmente de métodos polimórficos nas classes especializadas. Anteriormente, nesta mesma unidade, já foi explicado o conceito de herança e polimorfismo.

MULTIPLICIDADE

De acordo com Guedes (2007, p. 54), a multiplicidade determina o número mínimo e máximo de instâncias envolvidas em cada uma das extremidades da associação, permitindo também especificar o nível de dependência de um objeto para com os outros.

Quando não existe multiplicidade explícita, entende-se que a multiplicidade é “1..1”, significando que uma e somente uma instância dessa extremidade da associação relaciona-se com as

Anotações

instâncias da outra extremidade. A tabela abaixo, retirada de Guedes (2007, p. 55) demonstra alguns dos diversos valores de multiplicidade que podem ser utilizados em uma associação.

Tabela — Exemplos de multiplicidade

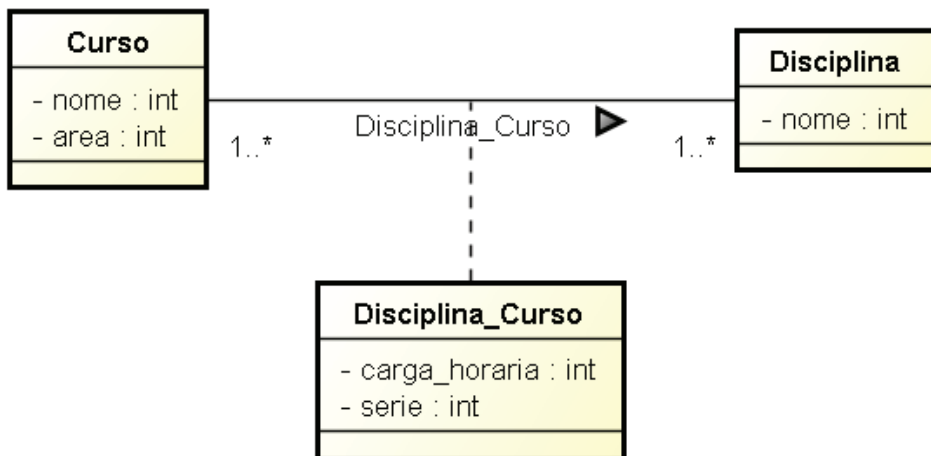
Multiplicidade	Significado
0..1	No mínimo zero (nenhum) e no máximo um. Indica que os objetos das classes associadas não precisam obrigatoriamente estar relacionados, mas se houver relacionamento indica que apenas uma instância da classe se relaciona com as instâncias da outra classe.
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.
0..*	No mínimo nenhum e no máximo muitos. Indica que pode ou não haver instâncias da classe participando do relacionamento.
*	Muitos. Indica que muitos objetos da classe estão envolvidos no relacionamento.
1..*	No mínimo 1 e no máximo muitos. Indica que há pelo menos um objeto envolvido no relacionamento, podendo haver muitos objetos envolvidos.
2..5	No mínimo 2 e no máximo 5. Indica que existem pelo menos 2 instâncias envolvidas no relacionamento e que podem ser 3, 4 ou 5 as instâncias envolvidas, mas não mais do que isso.

As associações que possuem multiplicidade do tipo muitos (*) em qualquer de suas extremidades, forçam a transmissão do atributo-chave contido na classe da outra extremidade da associação, porém esse atributo-chave não aparece desenhado na classe.

CLASSE ASSOCIATIVA

Quando um relacionamento possui multiplicidade muitos (*) em suas duas extremidades é necessário criar uma classe associativa para guardar os objetos envolvidos nessa associação. Na classe associativa são definidos o conjunto de atributos que participam da associação e não às classes que participam do relacionamento. Neste caso, pelo menos os atributos-chave devem fazer parte da classe associativa, porém, a classe associativa também pode possuir atributos próprios.

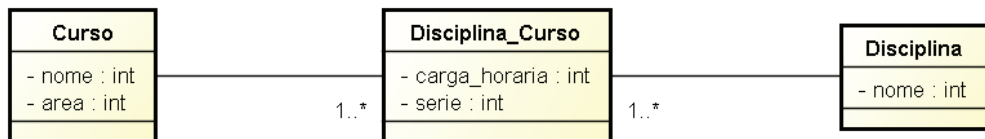
Uma classe associativa é representada por uma reta tracejada partindo do meio da associação e atingindo uma classe. A figura abaixo apresenta um exemplo de classe associativa que

possui atributos próprios, além dos atributos-chave das classes que participam da associação (só que nesse caso esses atributos-chave não são representados no diagrama).

Exemplo de classe associativa

Neste exemplo, uma instância da classe Curso pode se relacionar com muitas instâncias da classe Disciplina, e uma instância da classe Disciplina pode se relacionar com muitas instâncias da classe Curso. Como existe a multiplicidade muitos nas extremidades de ambas as classes da associação, não há como reservar atributos para armazenar as informações decorrentes da associação, pois não há como determinar um limite para a quantidade de disciplinas que um curso pode ter e nem há como saber em quantos cursos uma disciplina pode pertencer. Assim, é preciso criar uma classe para guardar essa informação, além das informações próprias da classe, que são a carga horária da disciplina no curso e também a qual série essa disciplina estará vinculada naquele curso. Os atributos-chave das classes Curso e Disciplina são transmitidos de forma transparente pela associação, não sendo, portanto, necessário escrevê-los como atributos da classe associativa.

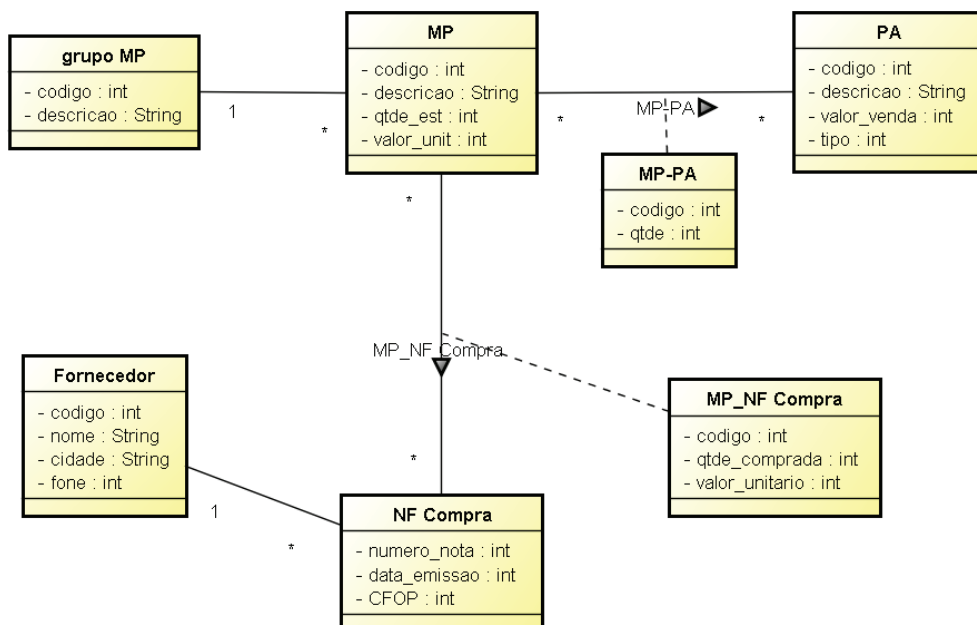
Segundo Guedes (2007, p. 61), as classes associativas podem ser substituídas por classes normais, que, neste caso, são chamadas de classes intermediárias, mas que desempenham

Anotações

exatamente a mesma função das classes associativas. A figura abaixo mostra o mesmo exemplo da figura anterior, porém com a classe intermediária ao invés da classe associativa.

Exemplo de classe intermediária

ESTUDO DE CASO

Para mostrar um diagrama de classes, vamos utilizar o mesmo documento de requisitos utilizado para demonstrar o diagrama de casos de uso, ou seja, o documento de requisitos da Fábrica de Colchões. Não vou colocar novamente o documento aqui, mas seria interessante que você fizesse a leitura novamente do documento antes de analisar o diagrama de classes abaixo.

Anotações

ESTUDO DE CASO 2

Neste estudo de caso, vou mostrar um novo documento de requisitos, agora de um Laboratório de Análises Clínicas. Com base nesse documento de requisitos, vamos elaborar o diagrama de casos de uso e também o diagrama de classes, mas dessa vez faremos isso passo a passo.

Mas antes de começarmos a ler o documento de requisitos, gostaria que você imaginasse que foi a um médico, por exemplo, um clínico geral. Para te dar um diagnóstico preciso e correto, esse médico pediu que você fizesse uma série de exames, como, por exemplo: hemograma, glicemia, creatinina, triglicerídeos e urocultura. Ele anotou essa lista de exames em um Pedido de Exames e você, de posse desse pedido, vai até um laboratório de análises clínicas para fazer os exames (nesse caso, você vai até o Laboratório São João). É aí que começa o nosso documento de requisitos.

Anotações

DOCUMENTO DE REQUISITOS – Laboratório de Análises Clínicas

O Laboratório de Análises Clínicas São João deseja informatizar suas atividades, pois hoje não há controle sobre o histórico de cada paciente (dos exames que ele já realizou) e se gasta muito tempo com atividades que poderiam ser feitas por um sistema informatizado.

Hoje, o laboratório funciona da seguinte forma:

- O paciente (você) chega ao laboratório com o pedido dos exames (preenchido pelo seu médico – aquele que o clínico geral que você acabou de consultar, preencheu).
- Se for a primeira vez que o paciente vai fazer exames, é preenchida uma ficha de cadastro com os seguintes dados: nome, endereço, cidade, UF, CEP, telefone, data de nascimento, RG e CPF.
- A recepcionista (a moça que te atendeu quando você chegou ao laboratório São João) preenche uma ficha com o nome do paciente, nome do convênio do paciente, os nomes dos exames que o paciente irá fazer e a data e horário da realização de cada exame. No final da ficha é anotada a data em que os exames estarão prontos. A primeira via desta ficha é entregue ao paciente (a você).
- O resultado do exame é colocado no envelope para posterior retirada pelo paciente.

O Laboratório deseja que o novo sistema possa fornecer informações rápidas, precisas e seguras, para assim melhorar suas atividades administrativas e o atendimento a seus pacientes (e neste caso, você vai demorar bem menos no laboratório, pois os processos estarão automatizados). Para tanto, o novo sistema deve:

- Permitir o cadastro dos pacientes do laboratório, com todos os dados preenchidos hoje na ficha de cadastro. Esse cadastro será realizado pelas recepcionistas.
- Permitir o cadastro dos exames que o laboratório pode realizar. Cada exame pertence a um Grupo de Exames. Por exemplo, o exame HEMOGRAMA, pertence ao grupo SANGUE. Para cada exame é preciso saber o seu código, descrição, valor e procedimentos para a realização do mesmo. Por exemplo, hemograma: deve estar em jejum. Esse cadastro será realizado pelos Bioquímicos.
- Permitir o cadastro dos pedidos de exames dos pacientes. É necessário saber qual o nome do paciente, o nome do médico que está solicitando os exames, o nome

Anotações

do convênio que o paciente irá utilizar para este pedido, data e horário dos exames (**atenção: cada exame pode ser realizado em datas e horários diferentes**), os nomes dos exames a serem feitos, data e horário em que cada exame ficará pronto (**atenção: cada exame pode ficar pronto em uma data e horário diferente**) e o valor de cada um deles. **Lembrando que o médico pode solicitar mais de um exame em cada pedido** (no seu caso, o médico solicitou 5 exames. Está lembrado do começo do nosso estudo de caso?). Esse cadastro será realizado pelas recepcionistas.

- Emitir a ficha do paciente, contendo todos os dados cadastrados. Este relatório será solicitado e recebido tanto pelas recepcionistas quanto pelo departamento administrativo do laboratório.
- Emitir relatório com todos os exames que o laboratório realiza com o código, descrição, procedimentos e valor de cada exame, agrupados por grupo de exame, devendo ser impresso neste relatório o código e a descrição do grupo. Este relatório será solicitado e recebido pelas recepcionistas.
- Emitir o pedido do exame em 3 vias, com todos os dados do pedido do exame. O relatório será emitido pela recepcionista, sendo que a 1ª via será entregue ao paciente (comprovante da entrega do exame), a 2ª via será entregue para o departamento de faturamento (para a cobrança dos exames dos convênios) e a 3ª via para os bioquímicos (para a realização dos exames).
- Emitir relatório com os resultados dos exames por pedido de exame, contendo o nome do paciente, data e horário do exame (da realização do exame), nome do médico que solicitou o exame, nome do convênio, o nome e o resultado de cada exame realizado, no caso de ter sido mais de um. Esse relatório será solicitado pela recepcionista e entregue ao paciente (não é necessário que a recepcionista fique com esse relatório).

PASSO A PASSO PARA A ELABORAÇÃO DO DIAGRAMA DE CASOS DE USO

1. Leia novamente o Documento de Requisitos acima e verifique quais são os usuários do sistema. Essas pessoas serão os atores. Faça uma lista com os atores.
   a. Recepcionista

Anotações

____

____

____

____

a. Bioquímicos
   b. Departamento Administrativo
   c. Departamento de Faturamento
   d. Paciente

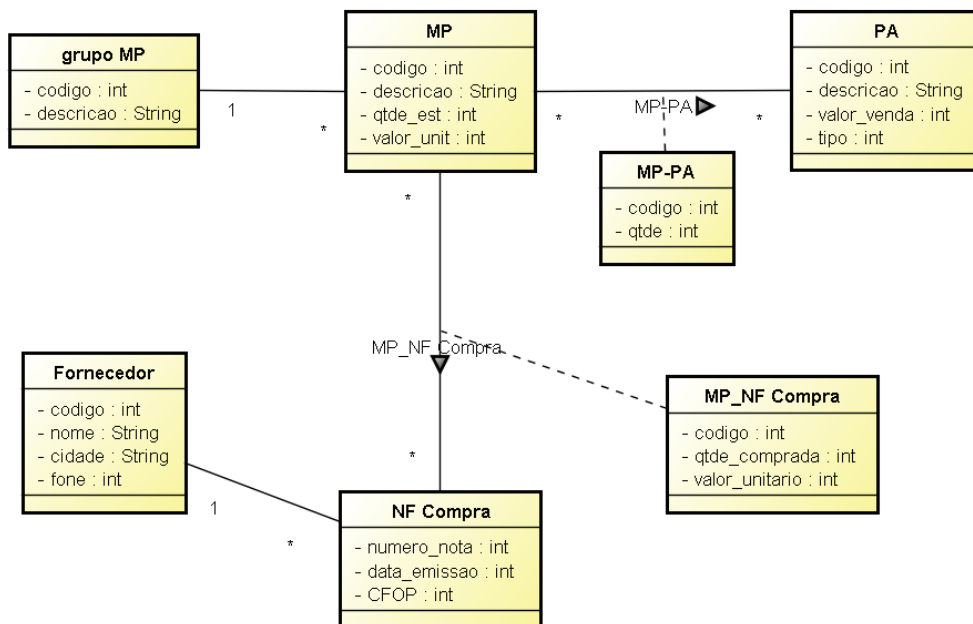
2. Agora faça uma lista com as funcionalidades do sistema. Algumas aparecem descritas claramente no documento de requisitos. Cada relatório que é mencionado no documento será uma funcionalidade. Então já sabemos que teremos as seguintes funcionalidades:
   a. Emitir ficha do paciente
   b. Emitir relatório de exames
   c. Emitir pedido de exame
   d. Emitir resultados dos exames

3. Porém, é importante observar que, além dos relatórios, um sistema precisa ter os cadastros para que o usuário consiga inserir os dados no sistema, para daí ser possível emitir os relatórios. Então, com base nos relatórios mencionados acima, teremos os seguintes cadastros:
   a. Cadastrar pacientes
   b. Cadastrar exames
   c. Cadastrar pedidos de exame
   d. Cadastrar resultados dos exames

4. Cada uma das funcionalidades relacionadas acima será um caso de uso em nosso diagrama. Então, agora você precisa verificar qual(is) ator(es) estará(ão) envolvido(s) em cada caso de uso. Isso você descobre fazendo uma nova leitura do documento de requisitos.

5. Além das funcionalidades já relacionadas, é importante pensar que algumas informações podem ser transformadas em classes, facilitando o uso e manutenção do sistema. Por exemplo: o sistema pode ter, além dos cadastros relacionados acima, um cadastro de médico, evitando que a recepcionista precisasse digitar o nome completo do médico toda vez que um pedido de exame daquele médico fosse lançado no sistema. Seguindo esse mesmo raciocínio alguns cadastros seriam importantes para o sistema:
   a. Cadastro de médicos
   b. Cadastro de convênios
   c. Cadastro de cidades
   d. Cadastro de UF´s

e. Cadastro de grupos de exames (neste caso esse cadastro é de suma importância, pois o relatório de exames deve ser agrupado por Grupo de Exames. Com a criação de um cadastro evita-se de o usuário cadastrar o mesmo grupo várias vezes).

6. Agora é só você utilizar a ferramenta Astah e desenhar o seu diagrama. Depois que você tiver terminado de desenhar o seu, veja se ficou parecido com o meu. Note que o nome do caso de uso sempre começa com um verbo.

PASSO A PASSO PARA A ELABORAÇÃO DO DIAGRAMA DE CLASSES

1. Como você já fez o diagrama de casos de uso vai ficar mais fácil elaborar o diagrama de classes. Com base nos casos de uso que você definiu, você vai fazer uma lista das possíveis classes do sistema. Lembre-se que os relatórios não serão classe por si só, mas para emitir cada relatório utilizaremos diversas classes. Uma dica: se o diagrama possui um caso de uso de cadastro, é certeza de que precisará de uma classe para armazenar os dados que serão cadastrados nesse caso de uso. Seguindo esse raciocínio teríamos as seguintes classes:

Anotações

a. Paciente

b. Exame

c. Pedido de Exame

d. Resultado de Exame

e. Médicos

f. Convênios

g. Cidades

h. UF´s

i. Grupos de Exames

2. Agora, para cada uma das classes listadas, relacione os possíveis atributos de cada uma delas. A maioria desses atributos já aparece descrita no documento de requisitos. Nunca se esqueça de voltar ao documento de requisitos sempre que tiver dúvidas.

   a. Paciente: código, nome, endereço, CEP, cidade, UF, telefone, data de nascimento, RG e CPF.

   b. Exame: código, descrição, valor, procedimentos, grup,o ao qual pertence o exame.

   c. Pedido de Exame: código, nome do paciente, nome do médico, nome do convênio, nomes dos exames que serão realizados, data e hora da realização de cada exame, data e hora em que cada exame ficará pronto, valor de cada exame.

   d. Resultado de Exame: descrição do resultado (para cada exame do pedido de exame o resultado deverá ser cadastrado).

   e. Médicos: CRM, nome (como o documento de requisitos não menciona nada sobre os dados dos médicos, coloquei somente os atributos que interessam para o pedido de exame).

   f. Convênios: código, nome (como o documento de requisitos não menciona nada sobre os dados dos convênios, coloquei somente os atributos que interessam para o pedido de exame).

   g. Cidades: código, nome, DDD (neste caso, mesmo o documento de requisitos não mencionando nada, esses atributos devem constar em qualquer classe de cidades).

   h. UF´s: sigla, nome.

   i. Grupos de Exames: código, descrição.

3. Desenhar as classes relacionadas acima, com seus respectivos atributos, no Diagrama de Classes. Faremos o desenho do diagrama também utilizando a ferramenta

Astah e no mesmo arquivo em que já desenhamos o diagrama de casos de uso. Assim, ficaremos com os dois diagramas em um único arquivo.

4. Para cada classe desenhada no diagrama, estabelecer o seu relacionamento com as demais classes. Lembre-se dos tipos de relacionamentos que estudamos: associação (unária e binária), generalização/especialização, agregação. Releia também a explicação sobre classes associativas.

5. Depois disso, estabelecer a multiplicidade de cada relacionamento, lembrando-se de eliminar atributos que podem ser obtidos por meio do relacionamento.

6. Primeiro desenhe o seu diagrama de classes e depois compare com o meu. Veja só como ficou o meu diagrama.

Paciente
- codigo : int
- nome : int
- endereço : int
- CEP : int
- telefone : int
- data_nascimento : int
- RG : int
- CPF : int

Médico
- CRM : int
- nome : int

Convênio
- codigo : int
- nome : int

1..*
0..*
1..*
0..*

Pedido Exame
- codigo : int

0..*

Cidade
- código : int
- nome : int
- DDD : int

Exame-Pedido Exame
- data_realização_exame : int
- hora_realização_exame : int
- data_pronto : int
- hora_pronto : int
- resultado : int
- valor : int

Exame-Pedido Exame

0..*
1..*

UF
- sigla : int
- nome : int

Exame
- código : int
- descrição : int
- valor : int
- procedimentos : int

1..*

Grupo Exame
- código : int
- descrição : int

Seguem alguns esclarecimentos sobre o diagrama acima:

Anotações

1. Um pedido de exame pode estar relacionado a somente um paciente, um médico e um convênio (no diagrama não aparece a multiplicidade 1, por ser o valor default de um relacionamento).
2. Um pedido de exame pode estar relacionado a um ou a vários exames (no caso desse documento de requisitos, a 5 exames).
3. Note que os atributos: data_realizacao_exame, hora_realização_exame, data_pronto, hora_pronto, resultado e valor estão armazenados na classe associativa que foi originada do relacionamento muitos para muitos entre Pedido de Exame e Exame. Isso se deve ao fato de que, para cada exame, esses atributos podem ser diferentes. Por exemplo: se o atributo DATA_PRONTO tivesse sido armazenado na classe PEDIDO_EXAME, seria possível cadastrar somente uma data em que os exames ficariam prontos. Mas, na realidade, não é isso o que acontece, ou seja, em uma lista de exames que o paciente precisa realizar pode-se ter exames que ficam prontos em 2 dias e outros que ficam prontos em 5 dias.
4. Veja que não foi criada uma classe RESULTADO_EXAME, pois como é somente uma descrição, decidiu-se armazená-la na classe associativa Exame-Pedido Exame.
5. Note também que na classe Pedido Exame não aparece o nome do paciente como relacionamos no item 2 desse Passo a Passo. Isto porque o nome será obtido por meio do relacionamento de Pedido Exame com Paciente. Não desenhamos o atributo-chave de paciente na classe Pedido Exame, mas ele está lá, ou seja, por meio dele é que buscaremos o nome do paciente na classe paciente quando precisarmos.
6. Ao invés de termos utilizado o recurso da classe associativa, poderíamos ter utilizado o relacionamento de agregação. Veja como ficaria (serão mostradas somente algumas classes, as demais não foram mostradas, pois ficariam iguais ao diagrama já mostrado anteriormente).

Anotações

1..*
0..*
0..*

Pedido Exame
- codigo : int

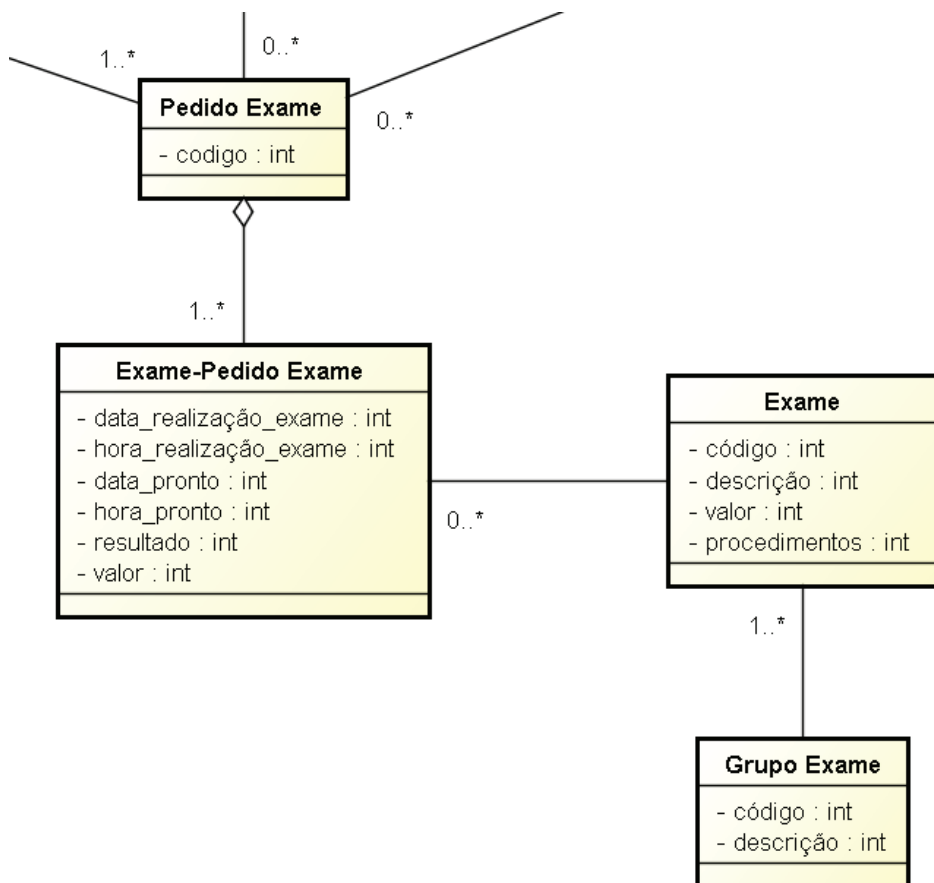
1..*

Exame-Pedido Exame
- data_realização_exame : int
- hora_realização_exame : int
- data_pronto : int
- hora_pronto : int
- resultado : int
- valor : int

0..*

Exame
- código : int
- descrição : int
- valor : int
- procedimentos : int

1..*

Grupo Exame
- código : int
- descrição : int

CONSIDERAÇÕES FINAIS

No decorrer desta unidade estudamos sobre a importância da modelagem de um sistema, a partir do documento de requisitos. A modelagem é a parte fundamental de todas as atividades, dentro de um processo de software, que levam à implantação de um bom software. É necessário construirmos modelos para comunicar a estrutura e o comportamento desejados do sistema, para melhor compreender o sistema que estamos elaborando (BOOCH, 2005, p. 3). Esses modelos, normalmente, são representados por meio de diagramas, em que é utilizada uma notação gráfica, que, em nosso caso, foi a UML.

A UML tem muitos tipos de diagramas, apoiando a criação de diferentes modelos de sistema, no entanto, conhecemos, nesta unidade, somente o Diagrama de Casos de Uso e o Diagrama de Classes, que são considerados por muitos autores, como os principais diagramas da UML.

A UML não estabelece uma ordem pré-definida para a elaboração dos seus diversos diagramas,

porém, como o diagrama de casos de uso, é um dos diagramas mais gerais e informais da UML, normalmente a modelagem do sistema inicia-se com a elaboração desse diagrama.

O diagrama de casos de uso mostra as interações entre um sistema e seu ambiente externo, determinando as funcionalidades e as características do sistema sob o ponto de vista do usuário. Conhecemos, nesta unidade, os conceitos necessários para a elaboração desse diagrama: atores, casos de uso e os possíveis relacionamentos entre estes elementos. Além disso, foi apresentado um diagrama pronto, que foi elaborado a partir do documento de requisitos para uma fábrica de colchões, mostrando como os conceitos acima podem ser aplicados em um diagrama.

Depois que o diagrama de casos de uso é elaborado fica bem mais fácil elaborar o diagrama de classes, que é o mais importante e também o mais utilizado da UML, e que define a estrutura das classes identificadas para o sistema, mostrando os atributos e métodos possuídos por cada uma das classes, e também os seus relacionamentos. Com base no mesmo documento de requisitos, o da fábrica de colchões, foi mostrado como fica o diagrama de classes referente ao mesmo.

E, para auxiliar o entendimento desses dois diagramas, foi apresentado a elaboração passo a passo de cada um deles. Para verificar se você realmente entendeu todos os conceitos, estou propondo mais um documento de requisitos nas atividades de autoestudo. Então mãos a obra!

ATIVIDADE DE AUTOESTUDO

1. Sobre relacionamento entre Casos de Uso e Atores
    a. Explique o relacionamento de Associação (*association*).
    b. Explique o relacionamento de Generalização entre casos de uso (*generalization*).
    c. Explique o relacionamento de Extensão (*extend*).
    d. Explique o relacionamento de Inclusão (*include*).
2. Explique qual a diferença entre os diagramas de classe abaixo:

Diagrama A

Gênero — 1..* Filme

Diagrama B

Gênero — 0..* Filme

3. Com base no documento de requisitos abaixo elaborar o Diagrama de Casos de Uso e o Diagrama de Classes.

DOCUMENTO DE REQUISITOS – CONTROLE DE ESTOQUE

A empresa Auto Peças XYZ Ltda atua no ramo de compra e venda de peças para veículos automotores. Atualmente, a empresa não possui sistema automatizado, mas pretende informatizar-se totalmente, para isso está contratando os serviços de um analista de sistemas. A empresa pretende começar pelo controle de estoque, pois o aumento do seu faturamento depende de um controle de estoque rígido.

O sistema atualmente funciona da seguinte maneira:

1. Os produtos são separados e controlados por grupos de produtos.
2. A empresa compra peças de diversos fornecedores. É preenchido um formulário denominado pedido de compra, no qual constam os seguintes dados: número do pedido, data do pedido, nome do fornecedor, telefone do fornecedor, nome do produto, quantidade do produto e preço unitário do produto. E, no final do pedido, uma totalização (quantidade * preço). Em cada pedido de compra tem-se somente um fornecedor, podendo ter-se vários produtos comprados.
3. Vende as peças à vista ou a prazo para os clientes. Nas vendas à vista são preenchidos os seguintes dados do cliente: nome, endereço completo, telefone e e-mail, para envio de correspondências (jornais de propaganda, ofertas). Nas vendas a prazo, são preenchidos os seguintes dados: nome, endereço completo, telefone, e-mail, RG, CPF, renda mensal, local de trabalho. Para cada venda é preenchida uma nota fiscal de venda, contendo os seguintes dados: número da nota fiscal, data da venda, nome do cliente, nome do vendedor, condição de pagamento, nome do produto, quantidade do produto e preço unitário do produto. Sendo que, no final da nota fiscal, é feito uma totalização da mesma (quantidade * preço unitário). Em cada nota fiscal de venda tem-se somente um cliente, podendo ter-se vários produtos vendidos.
4. Os vendedores são comissionados. Cada vendedor pode ter um % diferente de comissão sobre as vendas efetuadas. No final de cada mês, as vendas são somadas e a comissão do vendedor é calculada.

A empresa deseja que o sistema:

- Efetue o cadastro dos pedidos de compra que são preenchidos no item 2, atualizando automaticamente o estoque dos produtos que estão sendo comprados (somando no estoque). Este cadastro será realizado pelo Departamento de Compras.

- Efetue o cadastro das notas fiscais de vendas que são preenchidas no item 3, atualizando automaticamente o estoque do produto que está sendo vendido (diminuindo do estoque). Este cadastro será realizado pelo Departamento de Vendas.
- Calcule automaticamente as comissões dos vendedores de acordo com o % de cada um deles.
- Os demais cadastros serão efetuados pelo Departamento de Vendas.

O sistema deve emitir os seguintes relatórios:

- **Lista de preço de produtos por grupo de produtos.** Este relatório será solicitado e recebido pelo Departamento de Compras e pelo Departamento de Vendas.

Grupo: 99 - XXXXXXXXXXXXXXXXXXXX

Código Produto	Nome do Produto	Preço Unitário
999999	XXXXXXXXXXXXXXXXXXXXXXXX	999.999,99
999999	XXXXXXXXXXXXXXXXXXXXXXXX	999.999,99

- **Quantidade disponível em estoque por produto e grupo de produto.** Este relatório será solicitado e recebido pelo Departamento de Compras e pelo Departamento de Vendas.

Grupo: 99 - XXXXXXXXXXXXXXXXXXXX

Código Produto	Nome do Produto	Estoque Atual
999999	XXXXXXXXXXXXXXXXXXXXXXXX	9.999,99
999999	XXXXXXXXXXXXXXXXXXXXXXXX	9.999,99

- **Compras diárias – sintético.** Este relatório será solicitado e recebido pelo Departamento de Compras.

DATA: 99/99/9999

Nº PEDIDO	NOME FORNECEDOR	TOTAL COMPRADO
999999	XXXXXXXXXXXXXXXXXXXXXXXX	99.999.999,99
999999	XXXXXXXXXXXXXXXXXXXXXXXX	999.999.999,99

- **Vendas diárias por cliente – sintético.** Este relatório será solicitado e recebido pelo Departamento de Vendas.

DATA: 99/99/9999

Nº NF	NOME CLIENTE	VENDEDOR	TOTAL VENDIDO
999999	XXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXX	99.999.999,99
999999	XXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXX	99.999.999,99

- **Vendas por vendedor num determinado intervalo de data.** Este relatório será solicitado e recebido pelo Departamento de Vendas.

PERÍODO: 99/99/9999 A 99/99/9999

VENDEDOR: XXXXXXXXXXXXXXXXXXXXXXXXXXX

DATA	Nº NF	NOME CLIENTE	TOTAL VENDIDO	VLR COMIS-SAO
99/99/9999	999999	XXXXXXXXXXXXXX	9.999.999,99	999.999,99
99/99/9999	999999	XXXXXXXXXXXXXX	9.999.999,99	999.999,99
99/99/9999	999999	XXXXXXXXXXXXXX	9.999.999,99	999.999,99

UNIDADE V

PROJETO, TESTES E EVOLUÇÃO DE SOFTWARE

Professora Me. Márcia Cristina Dadalto Pascutti

Objetivos de Aprendizagem

- Expor a importância do projeto de software, mostrando os artefatos que serão criados durante o projeto.
- Estudar os formatos de entradas e saídas de informações através da interface do sistema.
- Entender o processo de validação de software, ou seja, quais os tipos de testes que devem ser realizados no sistema.
- Compreender a importância da evolução de software, para que o mesmo continue útil aos seus usuários.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- **Projeto de Software**
- **Diagrama Geral do Sistema, Interfaces e Especificação de Casos de Uso**
- **Testes de Software**
- **Evolução de Software**

INTRODUÇÃO

Na quarta unidade estudamos sobre a modelagem do sistema e aprendemos sobre Diagrama de Casos de Uso e Diagrama de Classes. Agora, para a próxima etapa do processo de software, ou seja, a etapa do projeto de software, precisaremos desses dois diagramas e também do documento de requisitos, para definir uma série de artefatos necessários para que, posteriormente, o software possa ser implementado.

A etapa de projeto de software, segundo Pressman (2011, p. 207), deve ser aplicada em qualquer modelo de processo de software que esteja sendo utilizado para o desenvolvimento do software e deve ser iniciada assim que os requisitos tiverem sido analisados e modelados, constituindo a última ação da modelagem e preparando a cena para a construção (geração de código e validação) do software.

O primeiro artefato de software que será explicado nesta unidade será o Diagrama Geral do Sistema (DGS). Neste diagrama devem aparecer todos os módulos e submódulos do sistema, assim como os itens que compõem cada um deles. O principal objetivo do DGS é mostrar como esses módulos e seus itens estão relacionados. Cada item que aparece no diagrama deve aparecer como um caso de uso no Diagrama de Casos de Uso.

Anotações

Depois que o DGS está definido, pode-se partir para a definição das interfaces com o usuário, tanto as de vídeo quanto as impressas. Para cada item que aparece no DGS, você deve pensar em como o usuário irá interagir com o sistema. Cada dado que deverá ser informado ao sistema, será por meio de uma interface que isso irá acontecer, portanto, quando a interface for definida, deverá ser verificado se os dados colocados na mesma estão definidos no Diagrama de Classes.

A interface do usuário é um dos elementos mais importantes de um sistema, e quando essa interface é mal projetada, a capacidade de o usuário aproveitar todo o potencial do sistema pode ser seriamente afetada, portanto, uma interface fraca pode fazer com que toda uma aplicação falhe (PRESSMAN, 2011, p. 313).

Nesta unidade, na seção Formatos de Entradas/Saídas, são mostradas algumas diretrizes importantes que o ajudarão a definir uma interface humana que seja de mais fácil compreensão para o usuário.

Depois que a interface foi definida e que o software foi implementado em alguma linguagem de programação, chegou a hora de executar a validação do software. A etapa de validação vai garantir que o software realmente executa as funcionalidades solicitadas pelos usuários.

E, finalmente, depois que o software foi validado e colocado em funcionamento, virá a etapa de evolução do software, pois, com certeza, os usuários terão requisitos que poderão ser alterados, implicando em alterações no software. Portanto, para que o software continue sendo útil para o usuário é necessário que ele evolua, atendendo as necessidades desse usuário.

PROJETO DE SOFTWARE

De acordo com Sommerville (2007, p. 50), um projeto de software é a descrição da estrutura de software a ser implementada, dos dados, das interfaces do sistema e, algumas vezes, dos

Anotações

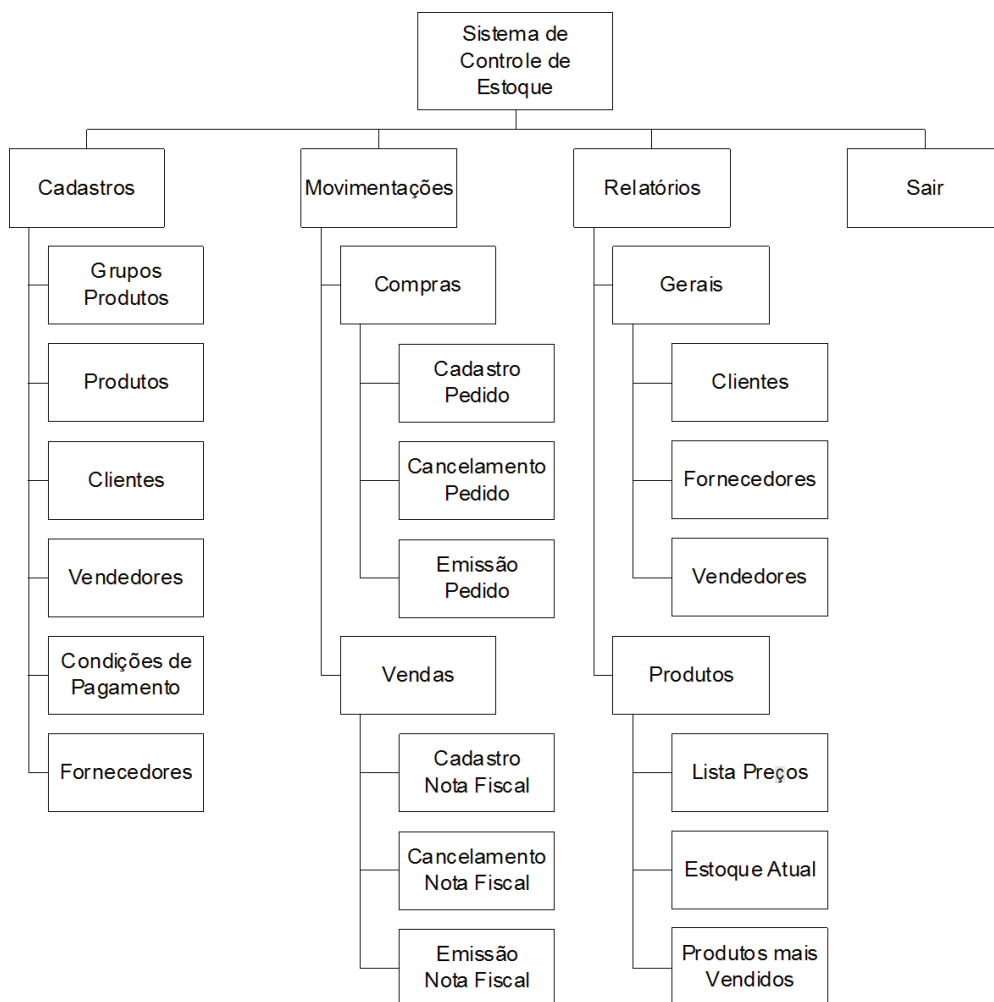
algoritmos utilizados (que, em nosso caso, será realizada por meio da Especificação dos casos de Uso). Um projeto deve ser desenvolvido de forma iterativa, por meio de diferentes versões que devem ser mostradas ao usuário para que ele ajude a decidir se o projeto realmente atende as suas necessidades.

Na etapa de modelagem do sistema, o engenheiro de software, com base no documento de requisitos, já elaborou o Diagrama de Caso de Uso e o Diagrama de Classes. Agora, na etapa de projeto, ele deverá: i) definir o Diagrama Geral do Sistema; ii) elaborar as Interfaces com o Usuário (telas e relatórios) e iii) desenvolver um conjunto de especificações de casos de uso, sendo que, essas especificações devem conter detalhes suficientes para permitir a codificação. Geralmente, durante o projeto, o engenheiro de software terá que definir cada componente do sistema ao nível de detalhamento que se fizer necessário para a sua implementação em uma determinada linguagem de programação.

Diagrama Geral do Sistema

Também conhecido por Diagrama de Módulos, apresenta os módulos do sistema, as ligações entre eles, os seus submódulos e/ou itens. O Diagrama Geral do Sistema deve ser condizente com o Diagrama de Caso de Uso, ou seja, as opções representadas no Diagrama Geral do Sistema devem aparecer como Casos de Uso no Diagrama de Casos de Uso. Seguem abaixo alguns exemplos de Diagrama Geral do Sistema.

Anotações

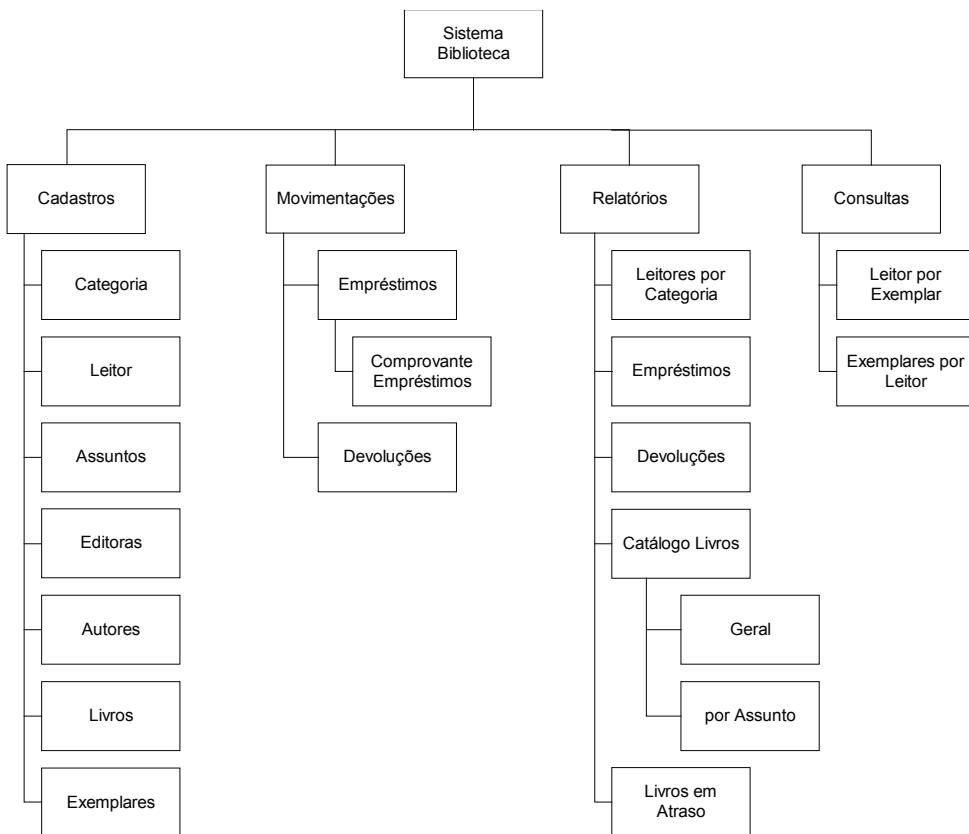
Exemplo 1 – Diagrama Geral do Sistema para um sistema de Controle de Estoque

- Sistema de Controle de Estoque
  - Cadastros
    - Grupos Produtos
    - Produtos
    - Clientes
    - Vendedores
    - Condições de Pagamento
    - Fornecedores
  - Movimentações
    - Compras
      - Cadastro Pedido
      - Cancelamento Pedido
      - Emissão Pedido
    - Vendas
      - Cadastro Nota Fiscal
      - Cancelamento Nota Fiscal
      - Emissão Nota Fiscal
  - Relatórios
    - Gerais
      - Clientes
      - Fornecedores
      - Vendedores
    - Produtos
      - Lista Preços
      - Estoque Atual
      - Produtos mais Vendidos
  - Sair

Fonte: a autora

O Diagrama de Caso de Uso referente a esse sistema deveria conter, pelo menos, os seguintes casos de uso: Cadastrar Grupos de Produtos, Cadastrar Produtos, Cadastrar Clientes, Cadastrar Vendedores, Cadastrar Condições de Pagamento, Cadastrar Fornecedores, Cadastrar Pedido

Anotações

de Compra, Efetuar Cancelamento de Pedido de Compra, Emitir Pedido de Compra, Cadastrar Nota Fiscal de Venda, Efetuar Cancelamento de Nota Fiscal de Venda, Emitir Nota Fiscal de Venda, Emitir Relatório de Clientes, Emitir Relatório de Fornecedores, Emitir Relatório de Vendedores, Emitir Lista de Preços de Produtos, Emitir Relatório de Estoque Atual de Produtos, Emitir Relatório de Produtos Mais Vendidos.

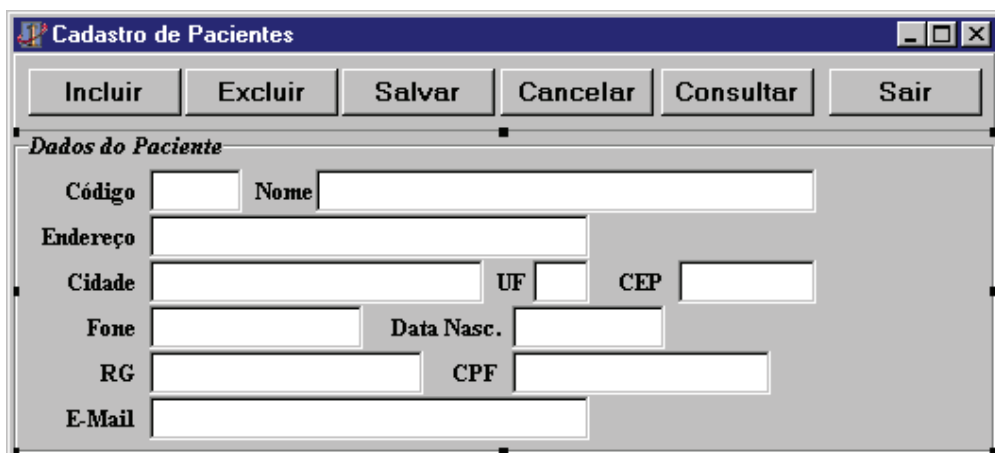
Exemplo 2 – Diagrama Geral do Sistema para um sistema de Controle de Biblioteca

- Sistema Biblioteca
  - Cadastros
    - Categoria
    - Leitor
    - Assuntos
    - Editoras
    - Autores
    - Livros
    - Exemplares
  - Movimentações
    - Empréstimos
      - Comprovante Empréstimos
    - Devoluções
  - Relatórios
    - Leitores por Categoria
    - Empréstimos
    - Devoluções
    - Catálogo Livros
      - Geral
      - por Assunto
    - Livros em Atraso
  - Consultas
    - Leitor por Exemplar
    - Exemplares por Leitor

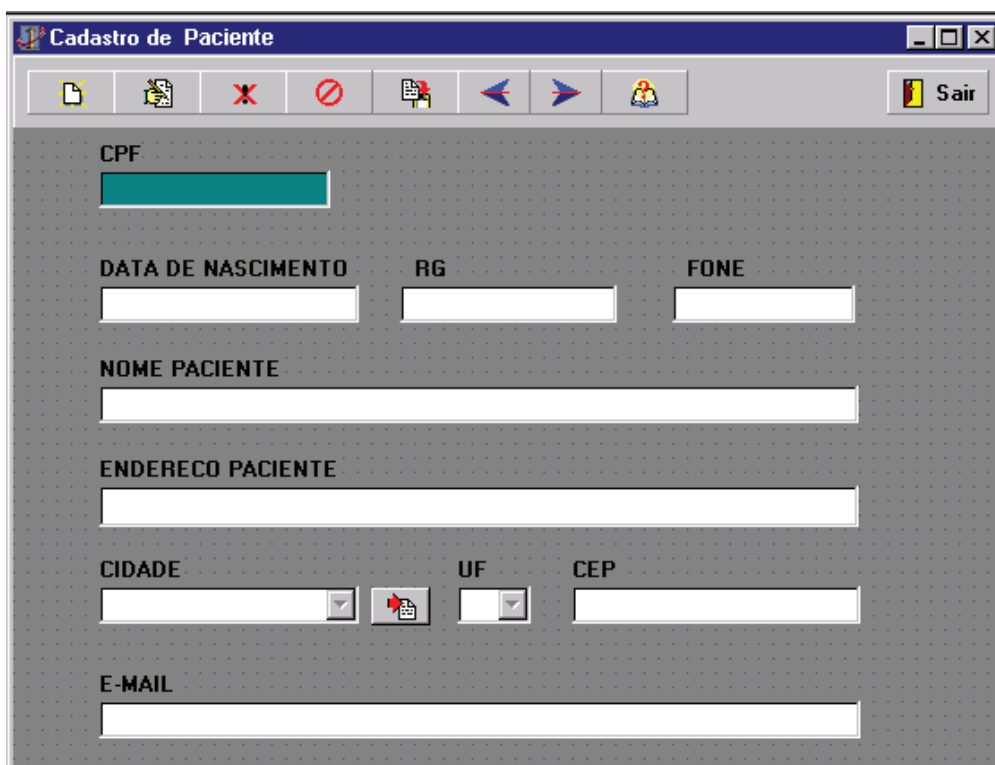
Fonte: a autora

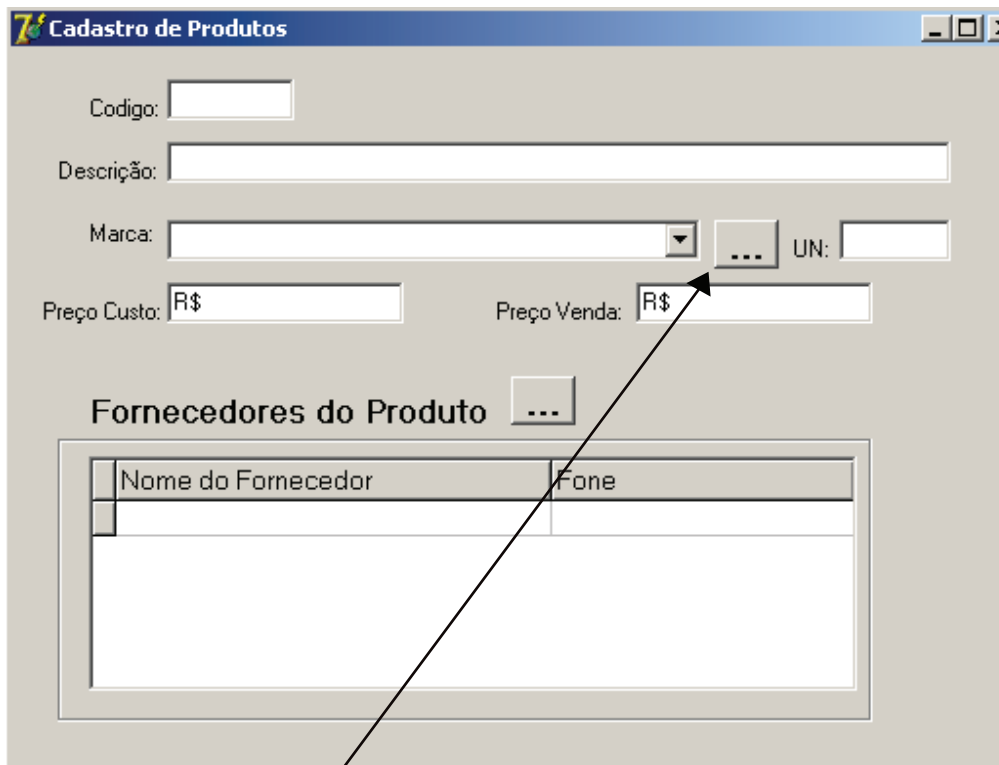
O Diagrama de Caso de Uso referente a esse sistema deveria conter, pelo menos, os seguintes

Anotações

casos de uso: Cadastrar Categorias, Cadastrar Leitores, Cadastrar Assuntos, Cadastrar Editoras, Cadastrar Autores, Cadastrar Livros, Cadastrar Exemplares, Registrar Empréstimos, Emitir Comprovante de Empréstimos, Registrar Devoluções, Emitir Relatório de Leitores por Categoria, Emitir Relatório de Empréstimos, Emitir Relatório de Devoluções, Emitir Catálogo Geral de Livros, Emitir Catálogo de Livros por Assunto, Emitir Relatório de Livros em Atraso, Efetuar Consulta de Leitor por Exemplar, Efetuar Consulta de Exemplares por Leitor.

Interfaces com o Usuário

Sommerville (2007, p. 241) comenta que o projeto de interface com o usuário deve ser feito cautelosamente, pois é uma parte fundamental de todo o processo de projeto de software. A interface com o usuário deve ser projetada para combinar as habilidades, experiências e expectativas dos usuários do sistema. A confiabilidade do sistema aumenta se um bom projeto de interface com o usuário for executado, se forem consideradas as capacidades dos usuários reais bem como o seu ambiente de trabalho. O engenheiro de software deverá definir todas as interfaces de vídeo e as interfaces impressas do sistema que está sendo projetado.

Interface de Vídeo

Para cada funcionalidade do sistema (caso de uso), definir o layout da tela. O layout é o desenho da tela e onde devem constar as informações que o usuário deverá fornecer ao sistema. Sugestão: criar uma padronização para todas as telas do sistema, para facilitar a utilização do mesmo pelo usuário. Seguem alguns exemplos de interfaces de vídeo.

- **Tela de Cadastro de Pacientes:** esta interface contém as informações pessoais de um paciente. Note que as informações encontram-se distribuídas de maneira lógica e o espaço foi otimizado para que em uma mesma tela várias informações pudessem ser colocadas.

Anotações

____

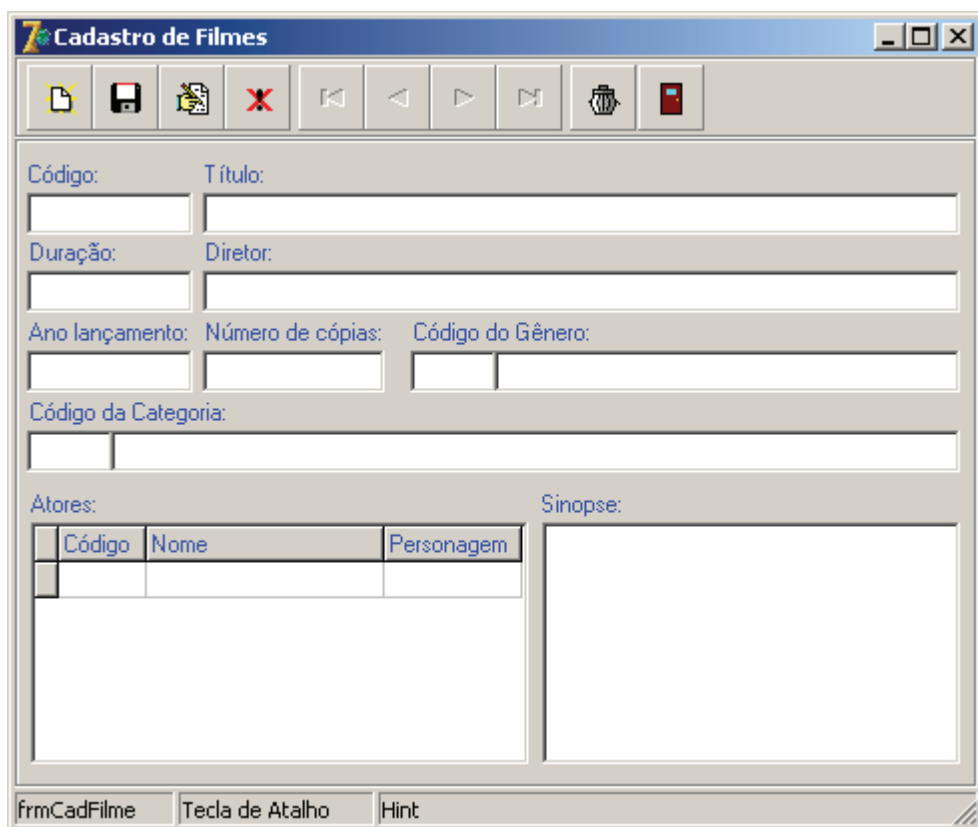
____

____

____

- **Outro Exemplo de Tela de Cadastro de Pacientes:** já nesta tela as informações não estão distribuídas de maneira lógica. Note que o nome do paciente somente é solicitado depois da data de nascimento, RG e telefone.

Cadastro de Paciente
Sair
CPF
DATA DE NASCIMENTO
RG
FONE
NOME PACIENTE
ENDERECO PACIENTE
CIDADE
UF
CEP
E-MAIL

- **Tela de Cadastro de Produtos:** nesta tela, além de cadastrar os dados do produto, também é possível cadastrar os fornecedores que fornecem o produto.

Este botão é um BOTÃO DE ATALHO, que chamará o cadastro de marcas, caso a marca de determinado produto não esteja cadastrada. Desta forma, o usuário não terá que sair do cadastro de produto, chamar o cadastro de marcas e depois chamar novamente a tela de cadastro de produtos.

- **Tela de Cadastro de Filmes:** nesta tela o usuário poderá informar os dados do filme, com seu gênero e categoria e todos os atores que fazem parte do elenco do filme. Note que os atores estão em uma grade, em que é possível cadastrar vários atores.

Anotações

Cadastro de Filmes

Código: Título:

Duração: Diretor:

Ano lançamento: Número de cópias: Código do Gênero:

Código da Categoria:

Atores: Sinopse:

Código	Nome	Personagem

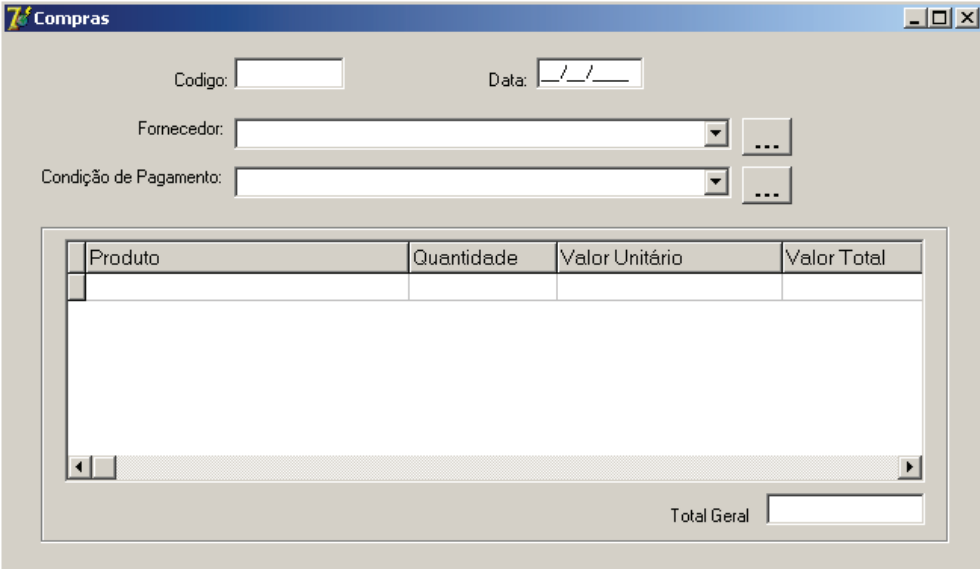
frmCadFilme | Tecla de Atalho | Hint

- **Tela de Venda:** nesta tela é possível lançar vários produtos em uma mesma venda, pois à medida que o lançamento do produto é efetuado e o usuário pressiona a tecla TAB, uma nova linha é acrescentada na grade de Produtos. O valor total de cada produto bem como o total geral da venda devem ser calculados automaticamente pelo sistema.

Anotações

- **Tela de Pedido de Exame:** nesta tela é possível lançar vários exames em um mesmo pedido de exame. Nesta mesma tela é possível, no caso do pedido de exame ser pago em parcelas, lançar o valor de cada parcela, o número do cheque deixado pelo paciente e a data de vencimento de cada parcela.

Pedido de Exame
Novo | Excluir | Cancelar | Salvar | Imprimir | Help
Nº do Pedido: 1
Paciente:
Médico:
Convênio:

Exame	Data da Realização	Hora da Realização	Data da Entrega	Hora da Entrega	Valor

Valor Total:

Parcela	Valor da Parcela	Nº do Cheque	Data do Vencimento

1 de 1
Sair

Anotações

Tela de Compras: nesta tela serão lançados os dados da compra, ou seja, a data da compra, o fornecedor, a condição de pagamento e os vários produtos que estão sendo comprados. Nesta tela também aparecem os botões de atalho. Note que o desenho do botão é o mesmo, assim o sistema fica padronizado e o usuário, quando se depara com esse botão em qualquer lugar do sistema, saberá que ao clicá-lo o sistema o levará para o cadastro referente ao dado em que o botão estava se referindo.

Compras

Codigo: Data: __/__/____

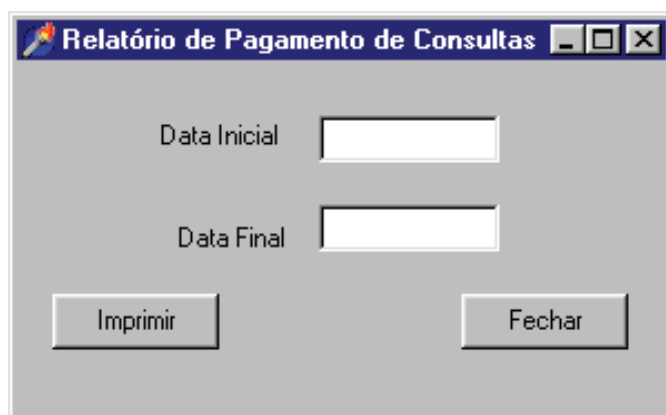
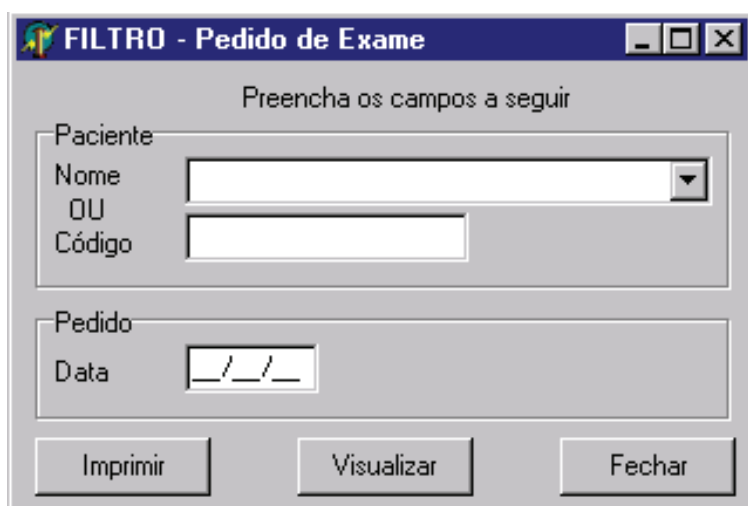
Fornecedor: ...

Condição de Pagamento: ...

Produto	Quantidade	Valor Unitário	Valor Total

Total Geral

Outro exemplo de Tela de Compras: nesta tela, além de lançar os produtos comprados, é possível já efetuar o lançamento do Contas a Pagar, ou seja, como na Nota Fiscal de Compra, normalmente já vem as informações da data de vencimento e do valor de cada parcela, quando uma compra é parcelada, o usuário já pode efetuar o lançamento dessas informações no momento em que está cadastrando as compras.

Anotações

- **Tela de Filtro do Relatório de Consultas Médicas Efetuadas por Dia:** a tela de filtro de relatório tem por objetivo oferecer ao usuário mecanismos para a seleção dos dados que ele deseja imprimir no relatório, pois, caso contrário, seriam impressos todos os dados cadastrados na base de dados referentes àquele relatório. Na tela abaixo o usuário irá informar uma data e o relatório emitirá somente as consultas efetuadas na data informada.

Anotações

- **Tela de Filtro do Relatório de Pagamento de Consultas por Período:** note que neste filtro de relatório o usuário poderá informar um intervalo de datas, dessa forma ele poderá emitir o relatório de um único dia, de uma semana, de um mês, ou seja, do período que ele desejar. Este tipo de filtro torna o relatório mais flexível. No exemplo abaixo serão emitidos os pagamentos das consultas efetuadas a partir da data inicial até a data final informadas na tela de filtro.

- **Tela de Filtro do Relatório de Pedido de Exame:** na tela abaixo são oferecidos dois filtros: primeiro o usuário deverá escolher se ele que selecionar por nome do paciente ou pelo seu código. Depois ele poderá escolher de qual data ele quer emitir o pedido de exame. As opções que deverão aparecer na tela de filtro dos relatórios deverão ser discutidas com o usuário, pois o objetivo é facilitar o andamento do seu trabalho quando o mesmo estiver utilizando o sistema.

FILTRO - Pedido de Exame

Preencha os campos a seguir

Paciente

Nome

OU

Código

Pedido

Data __/__/__

Imprimir Visualizar Fechar

- **Tela de Filtro do Relatório de Contas a Pagar por Fornecedor:** no exemplo abaixo o usuário poderá escolher, além do período, também o fornecedor que ele deseja emitir o relatório de contas a pagar.

B) Interface Impressa

Para cada relatório do sistema (definidos no Diagrama de Caso de Uso), é necessário definir o seu layout. Sugestão: criar uma padronização para todos os relatórios do sistema, para facilitar a utilização dos mesmos pelo usuário. Seguem alguns exemplos de interfaces impressas. Note nos exemplos que eles possuem no cabeçalho o nome da empresa, o número da página do relatório, a data (e em alguns o horário) da emissão do relatório, o título do relatório e, em alguns, o valor dos parâmetros informados na tela de filtro do relatório para que o usuário possa saber depois do relatório impresso, quais foram os dados informados no filtro para a emissão do mesmo.

- **Relatório de Clientes por Cidade**

NOME DA EMPRESA

DATA: 99/99/9999 Relatório de Clientes PAG.: 99

CÓDIGO	NOME	ENDERECO	CEP	TELEFONE
CIDADE: 9999 – XXXXXXXXXXXXXXXXXX				
9999	XXXXXXXXXXXXX	XXXXXXXXXXXXXXX	XXXXXXX	XXXXXXXXXXXX
9999	XXXXXXXXXXXXX	XXXXXXXXXXXXXXX	XXXXXXX	XXXXXXXXXXXX
9999	XXXXXXXXXXXXX	XXXXXXXXXXXXXXX	XXXXXXX	XXXXXXXXXXXX
9999	XXXXXXXXXXXXX	XXXXXXXXXXXXXXX	XXXXXXX	XXXXXXXXXXXX

- **Relatório de Consultas Efetuadas por Dia**

NOME DA EMPRESA		DATA:99/99/9999
Relatório de Consultas Efetuadas em 99/99/9999		PAG.: 99
PACIENTE	MÉDICO	CONVÊNIO
XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXX

- **Relatório de Pagamento de Consultas por Período**

NOME DA EMPRESA			99/99/9999 – 99:99
Relatório de Pagamentos de Consultas de 99/99/9999 a 99/99/9999 PAG.: 99			
DATA PAGAMENTO	PACIENTE	MÉDICO	VALOR CONSULTA
99/99/9999	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXX	999.999,99
99/99/9999	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXX	999.999,99
99/99/9999	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXX	999.999,99
99/99/9999	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXX	999.999,99
			VALOR TOTAL → 99.999.999,99

Especificação de Casos de Uso

A especificação de casos de uso descreve, por meio de uma linguagem bastante simples, as restrições que deverão ser consideradas no momento da implementação do caso de uso. Essas especificações devem conter detalhes suficientes para permitir a codificação em uma linguagem de programação qualquer. Seguem alguns exemplos de especificações de casos de uso.

- **Especificação de Caso de Uso – Cadastrar Paciente**
  - ◊ Os dados digitados pelo usuário deverão estar em caixa alta.
  - ◊ O código do paciente deverá ser gerado automaticamente pelo sistema, mas o usuá-

Anotações

rio poderá alterar. Nesse caso, não pode ser menor ou igual a zero.

◊ O nome do paciente não pode ser branco.

◊ Se CPF for digitado, aceitar somente CPF válido.

◊ Não permitir a exclusão de um paciente que tenha pelo menos um horário cadastrado na agenda.

- **Especificação de Caso de Uso – Cadastrar Agenda**

◊ Os dados digitados pelo usuário deverão estar em caixa alta.

◊ O código do paciente não pode ser menor ou igual a zero.

◊ Não permitir que seja marcada mais de uma consulta para o mesmo médico na mesma data e no mesmo horário.

◊ Quando for incluído um novo horário, mover “N” para o atributo CONS_REALIZADA.

◊ Quando a consulta for efetivada, mover “S” para CONS_REALIZADA.

◊ Se for consulta particular, pedir o valor da consulta.

◊ Data da consulta não pode ser menor que a data atual.

- **Especificação de Caso de Uso – Emitir Relatório de Pagamentos de Consultas por Período**

◊ Selecionar somente as instâncias da classe AGENDAS que tenham o atributo DATA maior ou igual à data inicial informada na tela de filtro do relatório e menor ou igual à data final e CONS_REALIZADA igual a “S” e VLR_CONSULTA diferente de zero.

◊ Não permitir data final menor que data inicial.

◊ Validar datas.

◊ Caso não exista nenhuma instância selecionada, mostrar mensagem de alerta “NÃO HÁ PAGAMENTO DE CONSULTAS NO PERÍODO INFORMADO” e não mostrar relatório em branco.

Anotações

FORMATOS DE ENTRADAS/SAÍDAS

Um dos problemas comuns de entrada/saída inclui a organização física de elementos de dados de entrada em uma tela de vídeo, a natureza das mensagens de erro quando se comete em erro de entrada e a organização física de elementos de dados de saída nas telas e em relatórios. Com o imenso conjunto atual de linguagens de quarta geração e ferramentas de prototipação, é interessante dar ao usuário diversas variações de telas de entrada, imagens de saída e coisas desse tipo. Uma vez obtido o consentimento do usuário, deve ser vinculada uma cópia das imagens de tela e formatos de relatórios à documentação do sistema.

É claro que em muitos casos o usuário não terá grandes sugestões para fazer porque não tem experiência anterior em trabalhar com um sistema informatizado. No caso dos sistemas de informações computadorizadas, as seguintes diretrizes ajudarão a desenvolver uma interface humana amistosa ao usuário na maioria dos casos.

1. Seu sistema deve requisitar entradas e produzir saídas em uma forma consistente. Isso é particularmente verdadeiro nos sistemas on-line em que o usuário pode introduzir uma entre diversas transações diferentes e/ou receber uma de diversas imagens diferentes. Por exemplo, se o sistema solicitar a data ao usuário sempre que uma transação for introduzida, seria desastroso se um tipo de transação exigisse a data na forma 12/23/87, enquanto outra transação exigisse a data na forma 87/12/23. E também seria desastroso se uma parte do sistema exigisse que o usuário especificasse o estado como um código de dois caracteres (p.ex., RJ para Rio de Janeiro), enquanto outra parte do sistema exigisse que o nome de estado fosse escrito por extenso. De forma análoga, se uma parte do sistema exigir que o usuário forneça um código de área quando introduzir um número de telefone, todas as partes do sistema devem exigir um código de área quando for introduzido um número de telefone.

2. Peça informações em uma sequência lógica. Em muitos casos, isso depende da natureza da aplicação e exigirá minuciosos entendimentos com o usuário. Por exemplo, se estiver sendo desenvolvido um sistema de controle de pedidos onde o usuário especifique o nome do cliente,

Anotações

será necessário especificar a sequência na qual os componentes de "nome de cliente" seriam introduzidos. Uma sequência lógica seria solicitar ao usuário que introduzisse a informação na seguinte sequência:

título (Sr./Sra. etc.)

primeiro nome

inicial do nome intermediário

último nome

Porém, o usuário pode achar isso muito desajeitado; suponha que o usuário tenha obtido o nome do cliente de alguma fonte externa como um catálogo de telefones. Nesse caso, seria muito mais prático que o usuário introduzisse.

último nome

inicial do nome intermediário

título

3. Evidencie ao usuário o tipo de erro que cometeu e onde está. Em muitos sistemas, o usuário fornece uma quantidade substancial de informações ao sistema para cada evento ou transação. Em um sistema de controle de pedidos, por exemplo, o usuário pode especificar o nome do cliente, endereço e número de telefone, bem como informações sobre itens que estão sendo pedidos, desconto aplicável, instruções de embarque e impostos de vendas. Todas essas informações podem ser introduzidas em uma única tela antes de serem gravadas; e, certamente, o sistema poderá determinar em seguida que parte da entrada está errada ou toda ela. O importante é ter certeza de que o usuário compreenda a espécie de erro que foi cometido e onde o erro está localizado; não é aceitável que o sistema simplesmente emita um sinal sonoro e exiba a mensagem MÁ ENTRADA. Cada erro (presumindo-se que haja mais de um) deve ser identificado, ou com uma mensagem descritiva ou (no caso de um sistema on-line) ressaltando ou codificando em cores os dados errados. Dependendo da natureza do sistema e do nível de sofisticação do usuário, pode ser também importante exibir uma

Anotações

mensagem explicativa.

4. Ofereça ao usuário um modo de (a) cancelar parte de uma transação e (b) cancelar toda uma transação. É ingênuo pressupor que o usuário sempre terminará a introdução de uma transação inteira sem ter interrompido. O usuário pode ter introduzido um nome e endereço de cliente antes de descobrir que estava trabalhando com o cliente errado; ele desejará ser capaz de anular o que foi digitado e começar de novo. Ou o usuário pode haver terminado a introdução da maior parte das informações do cliente e, então, perceber que escreveu mal o primeiro nome do cliente; ele vai querer poder retornar àquele elemento de dados e corrigi-lo sem perder todas as outras informações já digitadas.

5. Providencie um mecanismo prático de "socorro". Nos sistemas on-line é cada vez mais importante oferecer ao usuário um mecanismo prático para a obtenção de informações sobre como usar o sistema. Em alguns casos, a facilidade do "socorro" simplesmente fornecerá uma explicação se o usuário cometer um erro; em outros casos, o "socorro" poderia ser usado para explicar os detalhes de vários comandos ou transações disponíveis ao usuário. Existem muitos meios de implementar essa facilidade, e o analista e os usuários devem examinar diversos exemplos típicos antes de tomar uma decisão.

6. Se o seu sistema estiver empenhado em um processamento muito prolongado, exiba uma mensagem para que o usuário não pense que o sistema caiu. Se o seu sistema tiver de executar cálculos extensos ou se ele provavelmente apresentar demoras periódicas por causa de entradas volumosas, é importante exibir uma adequada mensagem para o usuário; caso contrário é possível que ele imagine que o seu computador "congelou" ou "ficou preso", ou que o sistema "caiu", ou que ocorreu uma falta de energia. O sistema deve, no mínimo, emitir uma mensagem (por exemplo, SISTEMA OPERANDO - POR FAVOR, ESPERE) em intervalos regulares. Melhor ainda seria uma série de mensagens informando o usuário quanto do trabalho já foi executado e aproximadamente quanto tempo falta para completá-lo, como acontece na maioria dos processos de instalação de aplicativos, como, por exemplo, Microsoft Word, Visio, Visual Studio.

Anotações

7. Forneça defaults para entradas padronizadas. Em muitos casos, o sistema pode fazer uma boa suposição sobre o provável valor de uma entrada fornecida pelo usuário; tornando-o um default, poderá ser economizado um considerável tempo do usuário. Essa abordagem é válida para todos os tipos de sistemas. Um exemplo de valor default é a data: em muitas aplicações a data mais provável que o usuário introduzirá é a data atual. Ou, suponha que você esteja construindo um sistema de controle de pedidos, e o usuário diz que 95% dos clientes vivem nas redondezas; em seguida, quando for solicitado que o usuário informe o número do telefone do cliente, o sistema deve presumir que o código de área default é o seu código de área local.

8. Beneficie-se das cores e do som, mas não exagere. Os efeitos de cores e som podem ser bastante úteis para a enfatização de tipos diferentes de entrada ou atrair a atenção do usuário para aspectos importantes da interface humana. Dessa forma, poderá ser utilizada a cor verde para tudo o que for exibido pelo sistema, azul para todos os dados de entrada fornecidos pelo usuário, e vermelho para todas as mensagens de erro. Entretanto, o analista não deve exagerar: a maioria dos usuários poderá não gostar se as telas ficarem parecendo árvores de Natal. Isso também vale para efeitos sonoros; uma ocasional campainha ou bip é útil, mas o usuário não deseja que o terminal produza todos os efeitos sonoros do filme Guerra nas Estrelas.

VALIDAÇÃO DE SOFTWARE

A validação de software tem como objetivo mostrar que um sistema está de acordo com as especificações descritas no documento de requisitos e que ele atende às expectativas do cliente comprador do sistema. Esse processo envolve verificar processos em cada estágio do processo de software, desde a definição dos requisitos dos usuários até o desenvolvimento de cada um dos programas que compõem o sistema. Porém, a maior parte dos custos de validação é observada depois da implementação, quando o sistema operacional é testado.

A atividade de testes é uma etapa muito importante para o desenvolvimento de software, normalmente inserindo tantos erros em um produto quanto a própria implementação. Por outro lado, caso um erro seja descoberto durante o desenvolvimento, o custo para sua correção será muito menor do que se esse mesmo erro tivesse sido descoberto somente na fase de manutenção.

O processo de testes pode ocupar grande parte do cronograma de desenvolvimento de sistema, dependendo do cuidado com que tenham sido executadas as atividades iniciais de especificação, projeto e implementação.

O que você precisa saber sobre testes como analista de sistemas? Em muitos casos, o analista de sistemas trabalha em estreita associação com o usuário para desenvolver um conjunto completo e abrangente de casos de testes. Esse processo de desenvolvimento de casos de testes de aceitação pode ser executado em paralelo com as atividades de implementação de projeto e programação, de modo que, quando os programadores tiverem terminado seus programas e executado seus próprios testes locais, a equipe usuário/analista estará pronta para seus próprios casos de testes (YOURDON, 1990, p.539).

TIPOS DE TESTES

Para Yourdon (1990, p. 540), existem diferentes estratégias de testes, porém as duas mais conhecidas são os testes *bottom-up* e os *top-down*. A abordagem *bottom-up* começa por

Anotações

testar os módulos pequenos de forma individual; essa modalidade é muitas vezes chamada de teste de unidade, teste de módulo ou teste de programa. Em seguida, os módulos individuais são agrupados com outros módulos para serem testados em conjunto; isso costuma ser chamado de teste de subsistemas. Finalmente, todos os módulos do sistema são combinados para um teste final, que é conhecido como teste do sistema, e é muitas vezes seguido pelo teste de aceitação, quando o usuário pode submeter dados reais para verificar se o sistema está funcionando corretamente.

A abordagem de testes *top-down* principia com um esqueleto do sistema, ou seja, a estratégia de testes presume que os módulos de execução de alto nível do sistema foram desenvolvidos, mas que os módulos de baixo nível existem apenas como simulações, somente como protótipos. Como a maioria das funções detalhadas do sistema não foi desenvolvida, os testes iniciais se tornam limitados, sendo possível apenas testar as interfaces entre os principais subsistemas. Os testes seguintes tornam-se em seguida cada vez mais abrangentes, testando aspectos cada vez mais detalhados do sistema.

Além desses conceitos básicos, você deve conhecer os seguintes tipos de testes:

- **Testes funcionais.** Neste tipo de teste o objetivo é verificar se o sistema executa corretamente suas funções para os quais ele foi projetado. Portanto, os casos de testes serão desenvolvidos e lançados no sistema; as saídas (e os resultados da atualização da base de dados) serão examinadas para testar sua correção (YOURDON, 1990, p. 540).

- **Testes de desempenho.** O objetivo deste tipo de teste é verificar se o sistema pode manipular o volume de dados e transações recebidas, bem como verificar se ele apresenta o tempo de resposta necessário (YOURDON, 1990, p. 541). Isso pode exigir que a equipe de desenvolvimento execute uma série de testes em que a carga do sistema é aumentada até que o seu desempenho se torne inaceitável (SOMMERVILLE, 2011, p. 153).

- **Testes de Unidade**. Têm por objetivo verificar um elemento que possa ser logicamente

Anotações

tratado como uma unidade de implementação, por exemplo, uma sub-rotina ou um módulo.

- **Testes de Aceitação.** Têm por objetivo validar o produto, ou seja, verificar se esse atende aos requisitos especificados. Eles são executados em ambiente o mais semelhante possível ao ambiente real de execução.

Se os casos de teste forem desenvolvidos no final da etapa de projeto, com utilização do diagrama de casos de uso, diagrama de classes e da especificação de casos de uso, não há meio de se saber como o programa funcionará quando o desenvolvedor escrever o código fonte e, desse modo, pode-se, então, executar o teste de caixa preta. Este tipo de teste focaliza os requisitos funcionais do software, determinando se os mesmos foram total ou parcialmente satisfeitos pelo produto. Os testes de caixa preta não verificam como ocorre o processamento, mas apenas os resultados produzidos. Pressman (2011, p. 439) coloca que o teste de caixa preta pode encontrar funções incorretas ou que estejam faltando, erros de interface, erros em estruturas de dados ou acesso a bases de dados externas, erros de comportamento ou de desempenho e, por último, erros de inicialização e término.

Quando a lógica e a estrutura interna do programa são conhecidas, isto é, depois de o programador ter escrito o programa, pode-se fundamentar os casos de testes na lógica do programa e executar o que muitas vezes é chamado de testes de caixa de vidro ou de caixa branca. Portanto, os testes de caixa branca pode, segundo Pressman (2011, p. 431), garantir que todos os caminhos independentes de um módulo foram exercidos pelo menos uma vez, exercitar todas as decisões lógicas nos seus estados verdadeiro e falso, executar todos os ciclos em seus limites e dentro de suas fronteiras operacionais e, por último, exercitar estruturas de dados internas para assegurar a sua validade.

Para Pressman (2011, p. 451), o teste dificilmente chega ao fim, o que acontece é uma transferência do desenvolvedor para o seu cliente, ou seja, toda vez que um cliente usa o sistema, um teste está sendo realizado.

Anotações

EVOLUÇÃO DE SOFTWARE

Após a implantação de um sistema é inevitável que ocorram mudanças, sejam elas para pequenos ajustes pós-implantação, para melhorias substanciais, por força da legislação, para atender novos requisitos dos usuários e finalmente, por estar gerando erros.

De acordo com Sommerville (2011, p. 164), cerca de dois terços de custos de software estão relacionados à evolução do software, requerendo grande parte do orçamento de Tecnologia da Informação para todas as empresas.

Manutenção de Software

O desafio da manutenção começa tão logo o software é colocado em funcionamento. Normalmente, depois que os usuários começam a utilizar o software, eles percebem que outras funcionalidades (ainda inexistentes) podem ser acrescentadas ao mesmo, ou seja, acabam aparecendo requisitos que o usuário não havia mencionado, pois foi com o uso do sistema que ele passou a perceber que o software pode oferecer mais do que está oferecendo. Como o sistema já está em funcionamento, essas novas funcionalidades são consideradas como manutenção.

Ou então a manutenção se dá para correção de erros no sistema, pois descobrir todos os erros enquanto o software está na etapa de validação é bastante complexo, pois todos os caminhos possíveis dentro dos programas teriam que ser testados e nem sempre isso é possível.

O fato é que a manutenção sempre vai existir e vai consumir bastante tempo da equipe de desenvolvimento. Pressman (2011, p. 663) coloca que "de fato, não é raro uma organização de software despender de 60% a 70% de todos os recursos com manutenção de software".

Uma das razões para o problema da manutenção de software é a troca das pessoas que compõem as equipes de desenvolvimento, podendo acontecer que a equipe que desenvolveu o software inicialmente, já não se encontra mais por perto. Ou ainda, que ninguém que esteja

Anotações

trabalhando atualmente na empresa, tenha participado da concepção do sistema. Neste caso, se o sistema desenvolvido estiver bem documentado, se ele tiver sido desenvolvido seguindo os preceitos da engenharia de software, com certeza, sua alteração se tornará mais fácil e pode-se dizer que o sistema apresenta alta manutenibilidade.

De acordo com Pressman (2011, p. 664), um software manutenível (de fácil manutenção) apresenta uma modularidade eficaz, utiliza padrões de projeto que permitem entendê-lo com facilidade, foi construído utilizando padrões e convenções de codificação bem definidos. Dessa forma, tanto o projeto quanto a implementação do software ajudarão a pessoa ou equipe que precisar realizar a alteração.

Sugestão de Vídeo

<http://www.youtube.com/watch?v=G6yk7fLK3JY&feature=relmfu>.
Vídeo que mostra a importância dos testes e da manutenção de um software.

CONSIDERAÇÕES FINAIS

Nesta última unidade, pudemos fechar todas as etapas do processo de software, ou seja, já tínhamos estudado a importância de definirmos bem os requisitos do sistema e deixar isso devidamente anotado em um documento de requisitos. Depois, estudamos sobre a modelagem do sistema e aprendemos a elaborar o diagrama de casos de uso e o diagrama de classes. E finalmente, estudamos sobre as etapas de projeto, validação e evolução de software.

A etapa de projeto de software se caracteriza por ser a definição física do sistema, ou seja, é onde podemos definir as interfaces do nosso sistema. Não entrei em muitos detalhes de como elaborar essa interface, pois este não é o nosso principal objetivo, mas se você tiver interesse pode estudar mais sobre o assunto Interação Humano-Computador (IHC) que é uma matéria interdisciplinar que relaciona a ciência da computação, artes, design, ergonomia, semiótica e outras áreas afins (veja só como esse assunto é abrangente!). Na etapa de projeto também é importante especificar cada caso de uso definido no diagrama de casos de uso. Você deve ter

notado que essa especificação vai ajudar, e muito, o programador a escrever o código fonte de cada programa, pois esta especificação deve conter detalhes sobre as restrições que cada programa deve considerar para que o sistema, como um todo, atinja o seu objetivo.

Depois que todos os artefatos descritos na modelagem e no projeto estiverem prontos, é hora da equipe de desenvolvimento codificar o sistema na linguagem de programação escolhida. Também não falamos nada sobre esse assunto, pois com certeza, você aprenderá (ou já aprendeu) as técnicas de programação em outras disciplinas do seu curso, e saberá que essa etapa é bastante trabalhosa e deve ser muito bem realizada para que todo o esforço despendido até aqui não seja em vão.

Depois que o software foi implementado, é hora da sua validação, ou seja, é hora de verificar se ele realmente está funcionando. Enquanto o sistema está sendo desenvolvido ele já está sendo testado pelas pessoas que o estão desenvolvendo, mas isto só não basta. É necessário desenvolver vários tipos de testes, como mostramos nesta unidade, para assegurar que o sistema funcionará corretamente. A real validação do software, normalmente, é feita quando o mesmo está em uso, pois é muito difícil testar todas as possibilidades de um sistema inteiro.

Após a implantação e efetiva utilização do sistema pelo usuário, qualquer alteração que seja necessária será considerada como uma evolução do software. Essa evolução é necessária para que o software continue sendo útil ao usuário, para que ele continue atendendo as suas necessidades. Se um software tiver uma vida longa, passará por manutenções durante esse período e, para que o mesmo continue manutenível, vimos que é necessário manter a aplicação das técnicas de engenharia de software, pois nem sempre quem desenvolve é quem vai dar manutenção no software.

Com isso, chegamos ao final das atividades básicas do processo de software. Espero que você tenha conseguido entender os conceitos relacionados a essas atividades, pois se você entendeu, conseguirá entender qualquer processo de software que possa vir a ser adotado pela empresa que você trabalha (ou trabalhará) como desenvolvedor(a).

ATIVIDADE DE AUTOESTUDO

1. Baseando-se no Documento de Requisitos – Laboratório de Análises Clínicas, mostrado na unidade IV – estudo de caso 2:
    a. Elabore o Diagrama Geral do Sistema.

b. Elabore as Interfaces de Vídeo dos seguintes casos de uso: Cadastrar exames, cadastrar médicos e cadastrar resultados de exame.

c. Elabore as telas de filtros dos relatórios de Ficha de Paciente e Relatório de Exame.

d. Elabore o layout dos relatórios de Ficha de Paciente e Exame.

2. Explique quatro diretrizes que devem ser levadas em consideração no desenvolvimento da interface do sistema.

3. Para que um sistema funcione corretamente é necessário que sejam efetuados vários testes. Explique como o analista deve conduzir esta etapa de testes, para que, no final, o sistema possa ser considerado apto a ser implantado.

CONCLUSÃO

Neste livro procurei mostrar a você a importância da disciplina de Engenharia de Software e como ela pode ser aplicada durante o desenvolvimento de um sistema. A fim de possibilitar o seu entendimento, na unidade I, foram estudados os conceitos de software e de engenharia de software. Mostrei também que podemos ter várias aplicações para o software, desde o software embutido que pode estar na sua máquina de lavar roupas até o software que controla um foguete espacial. Porém, neste material procurei utilizar exemplos que fazem parte do nosso dia a dia, pensando em facilitar o entendimento para problema e da sua possível solução.

Ainda na unidade I tratamos sobre ferramentas CASE, que são softwares que auxiliam o trabalho do desenvolvedor, automatizando tarefas que são realizadas durante o processo de software. Outro assunto que estudamos nesta unidade foram alguns conceitos de orientação a objetos e uma rápida introdução à linguagem de modelagem UML.

Na unidade II, trabalhamos especificamente com processos de software. Mostrei que podem existir diversos modelos de processos de software, mas que algumas atividades básicas estão presentes em todos eles (às vezes com nomes diferentes, mas estão presentes). Você está lembrado de quais são essas atividades? As atividades são: especificação de software, projeto e implementação de software, validação de software e, por último, evolução de software.

A unidade III foi dedicada exclusivamente a explicar o que são requisitos de software. Mostrei qual a diferença entre os requisitos funcionais e não funcionais e a importância do documento de requisitos, inclusive mostrando um exemplo.

Na unidade IV mostrei como, a partir do documento de requisitos, realizar a modelagem do sistema, utilizando a UML. Nesta unidade, expliquei com detalhes os Diagramas de Casos de Uso e Diagrama de Classes, dois dos mais importantes diagramas da UML. Apresentei também um exemplo de diagrama, partindo do documento de requisitos, explicando passo a passo a elaboração de cada um deles.

E para finalizar, vimos na unidade V, as etapas de projeto, validação e evolução de software, permitindo que você pudesse entender todas as etapas envolvidas nos modelos de processos de software.

Espero ter alcançado o objetivo inicial, que era mostrar a importância da Engenharia de Software.

Desejo que você seja muito feliz profissionalmente utilizando os conceitos apresentados aqui e se puder ajudar de alguma forma, estou a sua disposição. Desejo muito sucesso e paz.

Prof.ª Márcia.

REFERÊNCIAS

BOOCH, Grady. **UML**: guia do usuário. 2. ed. São Paulo: Campus, 2005.

GUEDES, Gilleanes T. A. **UML 2:** guia prático. São Paulo: Novatec Editora, 2007.

LIMA, Adilson da Silva. **UML 2.0:** do requisito à solução. São Paulo: Érica, 2009.

MELO, Ana Cristina. **Desenvolvendo Aplicações com UML 2.0:** do conceitual à implementação. Rio de Janeiro: Brasport, 2004.

PRESSMAN, Roger. **Engenharia de Software**. 7.ed. Porto Alegre: AMGH, 2011.

SILVA, Ricardo Pereira. **UML 2:** modelagem orientada a objetos . Florianópolis: Visual Books, 2007.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Addison-Wesley, 2007.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Prentice Hall, 2011.

YOURDON, Edward. **Análise Estruturada Moderna**. Rio de Janeiro: Elsevier, 1990.